



PILOT STRUCTURAL

Diseño de vigas de concreto reforzado.

CÓDIGO DEL PROGRAMA PILOT STRUCTURAL

Autor:

Juan David Garzón Obando

Coautores:

Ing. Juan David Tole Lozano

Ing. Janneth Patricia Gil

```

1 # Importaciones estándar
2 import math
3 from math import sqrt
4 import os
5 import re
6
7 # Importaciones de tkinter
8 import tkinter as tk
9 from tkinter import ttk, messagebox, filedialog, Toplevel, simpledialog
10 from tkinter import Scale, Frame, Entry, Label, NORMAL, DISABLED, END, HORIZONTAL
11 from tkinter import StringVar, PhotoImage
12 import time
13 import platform
14 from datetime import datetime
15 from pytz import timezone
16 import threading
17
18 # Importaciones de bibliotecas de terceros
19 from PIL import Image, ImageTk, ImageDraw
20 import numpy as np
21 import pandas as pd
22 import matplotlib.pyplot as plt
23 import matplotlib.gridspec as gridspec
24 from matplotlib import font_manager
25 from matplotlib.backends.backend_pdf import PdfPages
26 from matplotlib.ticker import MaxNLocator
27 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
28 from matplotlib.figure import Figure
29 import matplotlib.patches as patches
30 from mpl_toolkits.mplot3d.art3d import Poly3DCollection
31 from mpl_toolkits.mplot3d import Axes3D
32 from matplotlib.lines import Line2D
33 import scipy
34 from scipy.interpolate import CubicSpline
35 from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
36
37
38 # Importación de la librería de documentos de Word
39 from docx import Document
40 from docx.shared import Inches
41 import subprocess
42
43
44
45
46 "*****
47 "Autor: Juan David Garzón Obando. Programa: Ingeniería civil
48 Universidad: Universidad Piloto de Colombia"
49
50 "Proyecto: Desarrollo de un programa didáctico en Python para el análisis
51 estructural y diseño de vigas de concreto reforzado"
52
53 "*****
54 "*****ANÁLISIS ESTRUCTURAL Y DISEÑO DE VIGAS
55 "*****
56 "*****

```

```
54 # Función para mostrar el splash screen
55 def mostrar_splash():
56     splash = tk.Toplevel()
57     splash.overridereDIRECT(True)
58     splash.geometry("800x400")
59
60
61     splash.attributes('-topmost', True)
62
63     # Obtener el tamaño de la pantalla
64     screen_width = splash.winfo_screenwidth()
65     screen_height = splash.winfo_screenheight()
66
67     # Calcular las coordenadas para centrar la ventana
68     center_x = (screen_width - 800) // 2
69     center_y = (screen_height - 400) // 2
70
71     # Establecer la posición de la ventana en el centro de la pantalla
72     splash.geometry(f"800x400+{center_x}+{center_y}")
73
74     # Cargar y redimensionar la imagen del banner
75     banner_image = Image.open("Iconos_proyecto/Banner programa.png")
76     resized_banner_image = banner_image.resize((800, 400))
77     banner_image_tk = ImageTk.PhotoImage(resized_banner_image)
78
79     banner_label = tk.Label(splash, image=banner_image_tk)
80     banner_label.image = banner_image_tk
81     banner_label.pack()
82
83     # Crear un canvas para simular la barra de carga (color rojo)
84     progress_canvas = tk.Canvas(splash, width=400, height=30, bd=0,
highlightthickness=0, bg='white')
85     progress_canvas.place(x=70, y=250)
86
87     # Crear un rectángulo rojo dentro del canvas
88     progress_rect = progress_canvas.create_rectangle(0, 0, 0, 30, fill='red',
outline='')
89
90     # Crear un texto para el porcentaje dentro de la barra
91     progress_text = progress_canvas.create_text(200, 15, text="0%", font=
("Helvetica", 12, "bold"), fill="black")
92
93     # Función para actualizar la barra de progreso, el rectángulo rojo y el texto
del porcentaje
94     def actualizar_progreso():
95         for i in range(101): # 101 para llegar a 100%
96             progress_canvas.coords(progress_rect, 0, 0, i * 4, 30)
97             progress_canvas.itemconfig(progress_text, text=f"{i}%")
98             splash.update_idletasks()
99             time.sleep(0.05)
100     splash.destroy()
101     ventana.deiconify()
102     ventana.state('zoomed')
103
104     # Iniciar la actualización de la barra de progreso en un hilo para no bloquear
la ventana
105     threading.Thread(target=actualizar_progreso, daemon=True).start()
106
107     splash.mainloop()
108
```

```
109 # Variable global para controlar si la subventana ya está abierta
110 Ventana_Nuevo = None
111
112 # Crear la ventana principal
113 ventana = tk.Tk()
114
115 # Maximizar la ventana al iniciar
116 ventana.state('zoomed')
117
118 # Configuración de título
119 ventana.title("PILOT STRUCTURAL")
120
121 # Cargar y redimensionar el ícono usando PIL
122 ruta_icono = "Iconos_proyecto/Icono_P.png"
123 imagen = Image.open(ruta_icono)
124 imagen = imagen.resize((32, 32))
125 icono = ImageTk.PhotoImage(imagen)
126
127 # Establecer el ícono de la ventana
128 ventana.iconphoto(True, icono)
129
130 # Establecer el tamaño mínimo de la ventana
131 ventana.minsize(800, 600)
132
133
134 # Crear la barra de menús
135 menu_bar = tk.Menu(ventana)
136
137 # Definir iconos
138 icono_guardar = None
139 icono_abrir_archivo = None
140
141 # Definir funciones de guardar y abrir
142 def guardar_archivo():
143     # Definir el tipo de archivo a guardar
144     archivo = filedialog.asksaveasfilename(defaultextension=".pilot",
145                                             filetypes=[("Archivos de Pilot",
146                                                         "*.pilot"), ("Todos los archivos", "*.*")])
147     if archivo:
148         try:
149             # Guardar el contenido del archivo
150             with open(archivo, 'w') as f:
151                 # Aquí debes guardar el contenido relevante de tu aplicación
152                 f.write("Contenido de la aplicación...")
153             messagebox.showinfo("Guardado", f"Archivo guardado exitosamente: {archivo}")
154         except Exception as e:
155             messagebox.showerror("Error", f"No se pudo guardar el archivo: {e}")
156
157 def abrir_archivo():
158     # Definir los tipos de archivo que se pueden abrir (por ejemplo .miapp)
159     archivo = filedialog.askopenfilename(filetypes=[("Archivos de Pilot",
160                                                         "*.pilot"), ("Todos los archivos", "*.*")])
161     if archivo:
162         try:
163             # Abrir y leer el contenido del archivo
164             with open(archivo, 'r') as f:
165                 contenido = f.read()
166             # Aquí deberías cargar el contenido en tu aplicación
```

```
165         print(contenido) # Por ejemplo, mostrar el contenido en la
consola
166         messagebox.showinfo("Archivo abierto", f"Archivo abierto exitosamente:
{archivo}")
167         except Exception as e:
168             messagebox.showerror("Error", f"No se pudo abrir el archivo: {e}")
169
170
171
172 # Parámetros de la viga empotrada
173 base_viga = 0.40
174 altura_viga = 0.40
175 longitud_viga = 1
176 base_columna = 0.5
177 altura_columna = 1
178 longitud_columna = -0.2
179 desfase_estribo = 0.05
180
181 # Parámetros de la viga apoyada
182 base_viga_apoyada = 0.40
183 altura_viga_apoyada = 0.40
184 longitud_viga_apoyada = 1
185 base_piramide = -0.2
186 altura_piramdie = -0.5
187 longitud_piramide = -0.2
188 desfase_estribo = 0.05
189
190 # Función que se ejecuta al hacer clic en el botón
191 def on_boton_aplicar_click(label_contenido_1, label_prueba):
192     # Obtener el texto actual del label
193     tipo_viga = label_prueba.cget("text")
194
195     # Comprobamos si el texto de `label_prueba` tiene el valor esperado
196     if tipo_viga not in ["Viga apoyada", "Viga empotrada"]:
197         print("Texto en label_prueba no válido.")
198         return
199
200     # Eliminar solo los gráficos (por ejemplo, si hay un Canvas donde se dibujan
los gráficos)
201     for widget in label_contenido_1.wininfo_children():
202         if isinstance(widget, tk.Canvas): # Asegúrate de destruir solo el Canvas
203             widget.destroy()
204
205     # Si el label es "Viga apoyada"
206     if tipo_viga == "Viga apoyada":
207         # Mostrar gráfico de viga apoyada con sus parámetros
208         mostrar_grafico_viga_apoyada(
209             base_viga_apoyada,
210             altura_viga_apoyada,
211             longitud_viga_apoyada,
212             base_piramide,
213             altura_piramdie,
214             longitud_piramide,
215             desfase_estribo
216         )
217
218     # Si el label es "Viga empotrada"
219     elif tipo_viga == "Viga empotrada":
220         # Mostrar gráfico de viga empotrada con sus parámetros
221         mostrar_grafico_viga_empotrada(
```

```

222         base_viga,
223         altura_viga,
224         longitud_viga,
225         base_columna,
226         altura_columna,
227         longitud_columna,
228         desfase_estribo
229     )
230
231 # Grafico 3D viga empotrada
232 def dibujar_viga_empotrada(base, altura, longitud, posicion_x, posicion_y, color,
233 ax):
234     """
235     Dibuja un cuboide con las dimensiones especificadas y posición inicial.
236     """
237     # Definir las caras del cuboide
238     vertices = {
239         'cara_inferior': [
240             [posicion_x - base/2, posicion_x + base/2, posicion_x + base/2,
241             posicion_x - base/2],
242             [posicion_y, posicion_y, posicion_y + longitud, posicion_y +
243             longitud],
244             [0, 0, 0, 0]
245         ],
246         'cara_superior': [
247             [posicion_x - base/2, posicion_x + base/2, posicion_x + base/2,
248             posicion_x - base/2],
249             [posicion_y, posicion_y, posicion_y + longitud, posicion_y +
250             longitud],
251             [altura, altura, altura, altura]
252         ],
253         'cara_frental': [
254             [posicion_x - base/2, posicion_x + base/2, posicion_x + base/2,
255             posicion_x - base/2],
256             [posicion_y, posicion_y, posicion_y, posicion_y],
257             [0, 0, altura, altura]
258         ],
259         'cara_trasera': [
260             [posicion_x - base/2, posicion_x + base/2, posicion_x + base/2,
261             posicion_x - base/2],
262             [posicion_y + longitud, posicion_y + longitud, posicion_y + longitud,
263             posicion_y + longitud],
264             [0, 0, altura, altura]
265         ],
266         'cara_izquierda': [
267             [posicion_x - base/2, posicion_x - base/2, posicion_x - base/2,
268             posicion_x - base/2],
269             [posicion_y, posicion_y + longitud, posicion_y + longitud,
270             posicion_y],
271             [0, 0, altura, altura]
272         ],
273         'cara_derecha': [
274             [posicion_x + base/2, posicion_x + base/2, posicion_x + base/2,
275             posicion_x + base/2],
276             [posicion_y, posicion_y + longitud, posicion_y + longitud,
277             posicion_y],
278             [0, 0, altura, altura]
279         ]
280     }
281

```

```

270     # Dibujar cada cara del cuboide
271     for cara, coordenadas in vertices.items():
272         puntos = np.array(coordenadas).T # Transponer para formato adecuado
273         poligono = Poly3DCollection([puntos], color=color, edgecolor='black',
alpha=0.3)
274         ax.add_collection3d(poligono)
275
276 def dibujar_viga_y_columnas(base_viga, altura_viga, longitud_viga, base_columna,
altura_columna, longitud_columna, desfase_estribo):
277     """
278     Dibuja una viga con columnas en sus extremos.
279     """
280     fig = plt.figure(facecolor="white",dpi=100)
281     ax = fig.add_subplot(111, projection='3d')
282
283     # Dibuja la viga
284     dibujar_viga_empotrada(base_viga, altura_viga, longitud_viga, 0.5, 0, 'grey',
ax)
285
286     # Dibuja las columnas en los extremos de la viga
287     posicion_extremo_inicial = -longitud_columna # Columna en el extremo inicial
de la viga
288     posicion_extremo_final = longitud_viga # Columna en el extremo final de
la viga
289
290     dibujar_viga_empotrada(base_columna, altura_columna, longitud_columna, 0.5,
posicion_extremo_inicial, 'blue', ax) # Columna inicial
291     dibujar_viga_empotrada(base_columna, altura_columna, longitud_columna, 0.5,
posicion_extremo_final, 'blue', ax) # Columna final
292
293     # Ocultar etiquetas y ticks de los ejes sin afectar el dibujo 3D
294     ax.set_xticklabels([])
295     ax.set_yticklabels([])
296     ax.set_zticklabels([])
297     ax.set_xticks([])
298     ax.set_yticks([])
299     ax.set_zticks([])
300     ax.set_box_aspect([1, 3, -1])
301     ax.set_facecolor('white')
302     ax.xaxis.pane.fill = False
303     ax.yaxis.pane.fill = False
304     ax.zaxis.pane.fill = False
305     # Cambiar el color de las líneas del gráfico
306     ax.xaxis.pane.set_edgecolor('white')
307     ax.yaxis.pane.set_edgecolor('white')
308     ax.zaxis.pane.set_edgecolor('white')
309
310     return fig, ax
311
312 def mostrar_grafico_viga_empotrada(base_viga, altura_viga, longitud_viga,
base_columna, altura_columna, longitud_columna, desfase_estribo):
313
314     # Convertir la figura de matplotlib a un canvas de Tkinter
315     frame = tk.Frame(ventana,bd=10,relief="ridge")
316     frame.place(x=36,y=35,width=400,height=370)
317
318     # Crear la figura y el gráfico
319     fig, ax = dibujar_viga_y_columnas(base_viga, altura_viga, longitud_viga,
base_columna, altura_columna, longitud_columna, desfase_estribo)
320

```

```

321 # Integrar con Tkinter
322 canvas = FigureCanvasTkAgg(fig, master=frame)
323 canvas_widget = canvas.get_tk_widget()
324 canvas_widget.config(bg='white') # Fondo negro para el widget Canvas
325 canvas_widget.pack(fill=tk.BOTH, expand=True)
326
327 # Crear un frame para el gráfico y la barra de herramientas
328 frame_1 = tk.Frame(canvas_widget ,bd=2, relief="raise")
329 frame_1.place(x=0, y=315, width=380 ,height=35)
330
331 label_titulo = tk.Label(canvas_widget ,text="Modelo 3D - Viga
empotrada",bd=2,relief="raise")
332 label_titulo.place(x=1,y=1,width=380,height=35)
333
334
335 # Añadir la barra de herramientas de navegación
336 toolbar_opciones = NavigationToolbar2Tk(canvas, frame_1)
337 toolbar_opciones.update()
338 toolbar_opciones.pack(side=tk.BOTTOM, fill=tk.X)
339
340
341 # Función para dibujar la viga apoyada
342 def dibujar_viga_apoyada(base, altura, longitud, posicion_x, posicion_y, color,
ax):
343     """
344     Dibuja un cuboide con las dimensiones especificadas y posición inicial.
345     """
346     vertices = {
347         'cara_inferior': [
348             [posicion_x - base/2, posicion_x + base/2, posicion_x + base/2,
posicion_x - base/2],
349             [posicion_y, posicion_y, posicion_y + longitud, posicion_y +
longitud],
350             [0, 0, 0, 0]
351         ],
352         'cara_superior': [
353             [posicion_x - base/2, posicion_x + base/2, posicion_x + base/2,
posicion_x - base/2],
354             [posicion_y, posicion_y, posicion_y + longitud, posicion_y +
longitud],
355             [altura, altura, altura, altura]
356         ],
357         'cara_frontal': [
358             [posicion_x - base/2, posicion_x + base/2, posicion_x + base/2,
posicion_x - base/2],
359             [posicion_y, posicion_y, posicion_y, posicion_y],
360             [0, 0, altura, altura]
361         ],
362         'cara_trasera': [
363             [posicion_x - base/2, posicion_x + base/2, posicion_x + base/2,
posicion_x - base/2],
364             [posicion_y + longitud, posicion_y + longitud, posicion_y + longitud,
posicion_y + longitud],
365             [0, 0, altura, altura]
366         ],
367         'cara_izquierda': [
368             [posicion_x - base/2, posicion_x - base/2, posicion_x - base/2,
posicion_x - base/2],
369             [posicion_y, posicion_y + longitud, posicion_y + longitud,
posicion_y],

```

```

370         [0, 0, altura, altura]
371     ],
372     'cara_derecha': [
373         [posicion_x + base/2, posicion_x + base/2, posicion_x + base/2,
posicion_x + base/2],
374         [posicion_y, posicion_y + longitud, posicion_y + longitud,
posicion_y],
375         [0, 0, altura, altura]
376     ]
377 }
378
379 for cara, coordenadas in vertices.items():
380     puntos = np.array(coordenadas).T
381     poligono = Poly3DCollection([puntos], color=color, edgecolor='black',
alpha=0.3)
382     ax.add_collection3d(poligono)
383
384 # Función para dibujar las piramides
385 def dibujar_piramide(base, altura, posicion_x, posicion_y, color, ax):
386     """
387     Dibuja una pirámide invertida con base cuadrada.
388     """
389     # Definir los vértices de la pirámide invertida
390     vertices = [
391         [posicion_x - base / 2, posicion_y, altura], # Esquina inferior izquierda
de la base
392         [posicion_x + base / 2, posicion_y, altura], # Esquina inferior derecha
de la base
393         [posicion_x + base / 2, posicion_y + base, altura], # Esquina superior
derecha de la base
394         [posicion_x - base / 2, posicion_y + base, altura], # Esquina superior
izquierda de la base
395         [posicion_x, posicion_y + base / 2, 0] # Vértice superior (punta de la
pirámide)
396     ]
397
398     # Definir las caras de la pirámide
399     caras = [
400         [vertices[0], vertices[1], vertices[4]], # Cara frontal
401         [vertices[1], vertices[2], vertices[4]], # Cara derecha
402         [vertices[2], vertices[3], vertices[4]], # Cara trasera
403         [vertices[3], vertices[0], vertices[4]], # Cara izquierda
404         [vertices[0], vertices[1], vertices[2], vertices[3]] # Base
405     ]
406
407     # Dibujar cada cara de la pirámide
408     for cara in caras:
409         poligono = Poly3DCollection([cara], color=color, edgecolor='black',
alpha=0.3)
410         ax.add_collection3d(poligono)
411
412
413 # Función principal para dibujar la viga y las pirámides
414 def dibujar_viga_y_piramides(base_viga_apoyada, altura_viga_apoyada,
longitud_viga_apoyada, base_piramide, altura_piramdie, longitud_piramide,
desfase_estribo):
415     fig = plt.figure(facecolor="white", dpi=100)
416     ax = fig.add_subplot(111, projection='3d')
417
418     # Dibujar la viga

```

```

419     dibujar_viga_apoyada(base_viga_apoyada, altura_viga_apoyada,
longitud_viga_apoyada, 0.5, 0, 'grey', ax)
420
421     # Dibujar pirámides en los extremos de la viga
422     posicion_extremo_inicial = -longitud_piramide
423     posicion_extremo_final = longitud_viga_apoyada
424
425     dibujar_piramide(base_piramide, altura_piramdie, 0.5,
posicion_extremo_inicial, 'blue', ax) # Pirámide inicial
426     dibujar_piramide(base_piramide, altura_piramdie, 0.5, posicion_extremo_final,
'blue', ax) # Pirámide final
427
428     ax.set_xticklabels([])
429     ax.set_yticklabels([])
430     ax.set_zticklabels([])
431     ax.set_xticks([])
432     ax.set_yticks([])
433     ax.set_zticks([])
434     ax.set_box_aspect([1, 3, 1])
435     ax.set_facecolor('white')
436     ax.xaxis.pane.fill = False
437     ax.yaxis.pane.fill = False
438     ax.zaxis.pane.fill = False
439     ax.xaxis.pane.set_edgecolor('white')
440     ax.yaxis.pane.set_edgecolor('white')
441     ax.zaxis.pane.set_edgecolor('white')
442
443     return fig, ax
444
445 # Función para mostrar el gráfico en una ventana Tkinter
446 def mostrar_grafico_viga_apoyada(base_viga_apoyada, altura_viga_apoyada,
longitud_viga_apoyada, base_piramide, altura_piramdie, longitud_piramide,
desfase_estribo):
447     frame = tk.Frame(ventana, bd=10, relief="ridge")
448     frame.place(x=36, y=35, width=400, height=370)
449
450     fig, ax = dibujar_viga_y_piramides(base_viga_apoyada, altura_viga_apoyada,
longitud_viga_apoyada, base_piramide, altura_piramdie, longitud_piramide,
desfase_estribo)
451
452     canvas = FigureCanvasTkAgg(fig, master=frame)
453     canvas_widget = canvas.get_tk_widget()
454     canvas_widget.config(bg='white')
455     canvas_widget.pack(fill=tk.BOTH, expand=True)
456
457     frame_1 = tk.Frame(canvas_widget, bd=2, relief="raise")
458     frame_1.place(x=0, y=315, width=380, height=35)
459
460     label_titulo = tk.Label(canvas_widget ,text="Modelo 3D - Viga
apoyada",bd=2,relief="raise")
461     label_titulo.place(x=1,y=1,width=380,height=35)
462
463     toolbar_opciones = NavigationToolbar2Tk(canvas, frame_1)
464     toolbar_opciones.update()
465     toolbar_opciones.pack(side=tk.BOTTOM, fill=tk.X)
466
467
468 frame_ventana = tk.Frame(ventana, bd=10, relief="ridge",bg="white")
469 frame_ventana.place(x=36, y=35, width=400, height=370)
470

```

```
471
472 frame_1 = tk.Frame(frame_ventana, bd=2, relief="raise")
473 frame_1.place(x=0, y=315, width=380, height=35)
474
475 label_titulo = tk.Label(frame_ventana ,text="Modelo Viga 3D",bd=2,relief="raise")
476 label_titulo.place(x=1,y=1,width=380,height=35)
477
478
479
480 # Cargar las imágenes para los menús
481 imagen_original0 = Image.open("Iconos_proyecto/Nuevo_Modelo.png")
482 imagen_original_abrir=Image.open("Iconos_proyecto/Abrir_archivo.png")
483 imagen_original_guardar=Image.open("Iconos_proyecto/Disco_guardar.png")
484 imagen_original_salir= Image.open("Iconos_proyecto/Cerrar_sesion.png")
485 imagen_original = Image.open("Iconos_proyecto/Materiales.png")
486 imagen_original1 = Image.open("Iconos_proyecto/Seccion_transversal.png")
487 imagen_original2 = Image.open("Iconos_proyecto/Seccion_Viga.png")
488 imagen_original3= Image.open("Iconos_proyecto/apoyo_simple.png")
489 imagen_original4= Image.open("Iconos_proyecto/Cargas.png")
490 imagen_original5= Image.open("Iconos_proyecto/Grafico_V_M.png")
491 imagen_original6= Image.open("Iconos_proyecto/Flexion_VG.png")
492 imagen_original7= Image.open("Iconos_proyecto/Calculos_previos.png")
493 imagen_original8= Image.open("Iconos_proyecto/barra_de_acero.png")
494 imagen_original9= Image.open("Iconos_proyecto/viga_acero.png")
495 imagen_original10= Image.open("Iconos_proyecto/longitd_desarrollo.png")
496 imagen_original11= Image.open("Iconos_proyecto/Cortante.png")
497 imagen_original12 = Image.open("Iconos_proyecto/logo_programa.png")
498 imagen_original13 = Image.open("Iconos_proyecto/Doc_word.png")
499 imagen_original14 = Image.open("Iconos_proyecto/Doc_word_cortante.png")
500 imagen_original15 = Image.open("Iconos_proyecto/Model_3D.png")
501
502 # Redimensionar las imágenes (ajustar el tamaño según sea necesario)
503 imagen_redimensionada0 = imagen_original0.resize((20, 20))
504 imagen_redimensionada_abrir = imagen_original_abrir.resize((20, 20))
505 imagen_redimensionada_guardar= imagen_original_guardar.resize((20,20))
506 imagen_redimensionada_salir= imagen_original_salir.resize((20,20))
507 imagen_redimensionada = imagen_original.resize((20, 20))
508 imagen_redimensionada1 = imagen_original1.resize((20, 20))
509 imagen_redimensionada2 = imagen_original2.resize((20, 20))
510 imagen_redimensionada3 = imagen_original3.resize((20,20))
511 imagen_redimensionada4 = imagen_original4.resize((20,20))
512 imagen_redimensionada5 = imagen_original5.resize((20,20))
513 imagen_redimensionada6 = imagen_original6.resize((20,20))
514 imagen_redimensionada7 = imagen_original7.resize((20,20))
515 imagen_redimensionada8 = imagen_original8.resize((20,20))
516 imagen_redimensionada9 = imagen_original9.resize((20,20))
517 imagen_redimensionada10 = imagen_original10.resize((20,20))
518 imagen_redimensionada11 = imagen_original11.resize((20,20))
519 imagen_redimensionada12 = imagen_original12.resize((20,20))
520 imagen_redimensionada13 = imagen_original13.resize((20,20))
521 imagen_redimensionada14 = imagen_original14.resize((20,20))
522 imagen_redimensionada15 = imagen_original15.resize((20,20))
523
524
525 # Convertir las imágenes redimensionadas a un formato compatible con Tkinter
526 icono_modelo_nuevo = ImageTk.PhotoImage(imagen_redimensionada0)
527 icono_abrir_archivo = ImageTk.PhotoImage(imagen_redimensionada_abrir)
528 icono_guardar = ImageTk.PhotoImage(imagen_redimensionada_guardar)
529 icono_salir = ImageTk.PhotoImage(imagen_redimensionada_salir)
530 icono_materiales = ImageTk.PhotoImage(imagen_redimensionada)
```

```
531 icono_seccion_transversal = ImageTk.PhotoImage(imagen_redimensionada1)
532 icono_seccion_viga = ImageTk.PhotoImage(imagen_redimensionada2)
533 icono_asignar_apoyos = ImageTk.PhotoImage(imagen_redimensionada3)
534 icono_asignar_cargas = ImageTk.PhotoImage(imagen_redimensionada4)
535 icono_graficos = ImageTk.PhotoImage(imagen_redimensionada5)
536 icono_flexion = ImageTk.PhotoImage(imagen_redimensionada6)
537 icono_calculos = ImageTk.PhotoImage(imagen_redimensionada7)
538 icono_barras = ImageTk.PhotoImage(imagen_redimensionada8)
539 icono_VG_AS = ImageTk.PhotoImage(imagen_redimensionada9)
540 icono_longitud_desarrollo = ImageTk.PhotoImage(imagen_redimensionada10)
541 icono_cortante = ImageTk.PhotoImage(imagen_redimensionada11)
542 icono_pilot_structural = ImageTk.PhotoImage(imagen_redimensionada12)
543 icono_Doc_word = ImageTk.PhotoImage(imagen_redimensionada13)
544 icono_Doc_word_cortante = ImageTk.PhotoImage(imagen_redimensionada14)
545 icono_model_3D = ImageTk.PhotoImage(imagen_redimensionada15)
546
547
548 # Menú "Definir"
549 menu_definir = tk.Menu(menu_bar, tearoff=0)
550
551 # Menú "Archivo"
552 menu_archivo = tk.Menu(menu_bar, tearoff=0)
553 menu_archivo.add_command(label="Nuevo", image=icono_modelo_nuevo,
554     compound="left", command=lambda: abrir_ventana_nueva())
554 menu_archivo.add_command(label="Abrir", image=icono_abrir_archivo, compound="left",
555     command=abrir_archivo)
555 menu_archivo.add_command(label="Guardar", image=icono_guardar, compound="left", comma
556     nd=guardar_archivo)
556 menu_archivo.add_separator()
557 menu_archivo.add_command(label="Salir",
558     command=ventana.quit, image=icono_salir, compound="left")
558 menu_bar.add_cascade(label="Archivo", menu=menu_archivo)
559
560 # Declarar checkbutton_1 como una variable global
561 checkbutton_1 = None
562 checkbutton_2 = None
563 checkbutton_3 = None
564 checkbutton_4 = None
565
566 # Clase para manejar la reproducción del GIF
567 class GIFPlayer:
568     def __init__(self, master, gif_path, canvas, width, height):
569         self.master = master
570         self.canvas = canvas
571         self.gif_path = gif_path
572         self.width = width
573         self.height = height
574
575         # Verificar si el archivo existe antes de intentar cargarlo
576         if not os.path.exists(gif_path):
577             print(f"El archivo {gif_path} no existe.")
578             return
579
580         try:
581             self.image = Image.open(gif_path)
582         except Exception as e:
583             print(f"Error al abrir el archivo GIF: {e}")
584             return
585
586         self.frames = self.load_frames()
```

```

587         if not self.frames:
588             print("No se pudieron cargar los fotogramas del GIF.")
589             return
590
591         self.current_frame = 0
592         self.canvas_id = self.canvas.create_image(0, 0, anchor="nw",
image=self.frames[self.current_frame])
593         self.update_frame()
594
595     def load_frames(self):
596         """Carga los fotogramas del GIF"""
597         frames = []
598         try:
599             while True:
600                 # Escalar cada fotograma del GIF al tamaño del Canvas
601                 frame = self.image.copy()
602                 frame = frame.resize((self.width, self.height),
Image.Resampling.LANCZOS)
603                 frame = ImageTk.PhotoImage(frame)
604                 frames.append(frame)
605                 self.image.seek(self.image.tell() + 1)
606             except EOFError:
607                 pass
608             except Exception as e:
609                 print(f"Error al cargar los fotogramas del GIF: {e}")
610             return frames
611
612     def update_frame(self):
613         """Actualizar el fotograma a mostrar"""
614         self.current_frame = (self.current_frame + 1) % len(self.frames)
615         self.canvas.itemconfig(self.canvas_id,
image=self.frames[self.current_frame])
616         # Llamar a update_frame nuevamente para crear la animación continua
617         self.master.after(100, self.update_frame)
618
619 # Función para manejar el evento de selección en el Combobox
620 def on_combobox_select(event, canvas_viga, label_mensaje_ayuda, label_prueba):
621     selected_option = combobox.get().strip() # Eliminar cualquier espacio extra
622
623     # Lógica para actualizar el mensaje según la opción seleccionada
624     if selected_option == "Viga empotrada":
625         gif_path = os.path.join("Iconos_proyecto", "animacion.gif")
626         mensaje = "Has seleccionado la viga empotrada.\nPara el modelo sigue
aplicando las otras configuraciones.\nPara completar el modelo."
627     elif selected_option == "Viga apoyada":
628         gif_path = os.path.join("Iconos_proyecto", "viga_apoyada.gif")
629         mensaje = "Has seleccionado la viga apoyada.\nPara el modelo sigue
aplicando las otras configuraciones.\nPara completar el modelo."
630     else:
631         print("Opción no válida")
632         return
633
634     # Actualiza el texto del label de mensaje de ayuda
635     label_mensaje_ayuda.config(text=mensaje)
636
637     # Actualiza el texto en label_prueba con la opción seleccionada
638     label_prueba.config(text=f"{selected_option}")
639
640     # Verificar si el archivo GIF existe y mostrarlo en el canvas
641     if os.path.exists(gif_path):

```

```
642         if canvas_viga is not None:
643             gif_player = GIFPlayer(master=Ventana_Nuevo, gif_path=gif_path,
canvas=canvas_viga, width=190, height=170)
644         else:
645             print("Error: canvas_viga es None.")
646     else:
647         print(f"El archivo {gif_path} no existe.")
648
649 # Declarar label_prueba como variable global
650 label_prueba = None
651 subventana_abierta= None
652
653
654 # Ruta de la carpeta base_datos
655 carpeta_base_datos = "base_datos"
656 archivo_datos = os.path.join(carpeta_base_datos, "datos_modelo.txt")
657
658 # Crear la carpeta base_datos si no existe
659 if not os.path.exists(carpeta_base_datos):
660     os.makedirs(carpeta_base_datos)
661
662 def guardar_datos():
663     """Guarda los datos de los widgets en un archivo de texto."""
664     nombre_modelo = entry_nombre_modelo.get().strip()
665     nombre_usuario = entry_nombre_usuario.get().strip() # Obtenemos el nombre del
usuario
666
667     datos = {
668         "nombre_usuario": nombre_usuario,
669         "nombre_modelo": nombre_modelo,
670         "detalles_modelo": text_detalles_modelo.get("1.0", tk.END).strip(),
671         "combo_concreto": combo_concreto.get(),
672         "combo_acero": combo_acero.get()
673     }
674
675     # Guardar los datos en el archivo
676     with open(archivo_datos, "w") as archivo:
677         for clave, valor in datos.items():
678             archivo.write(f"{clave}:{valor}\n")
679
680     # Actualizar los Labels con el nombre del usuario y el nombre del modelo
681     label_estudiante.config(text=nombre_usuario) # Actualiza el texto del label
con el nombre del usuario
682     label_modelo.config(text=nombre_modelo) # Actualiza el texto del label con el
nombre del modelo
683
684     # Actualizar el título de la subventana
685     if nombre_modelo:
686         Ventana_Nuevo.title(f"Nuevo Modelo - {nombre_modelo}")
687
688     # Cerrar la ventana
689     Ventana_Nuevo.destroy()
690
691
692 def cargar_datos():
693     """Carga los datos desde un archivo de texto si existe."""
694     datos = {}
695     try:
696         with open(archivo_datos, "r") as archivo:
697             for linea in archivo:
```

```
698         clave, valor = linea.strip().split(":", 1)
699         datos[clave] = valor
700     except FileNotFoundError:
701         pass
702     return datos
703
704
705 # Función que se ejecuta cuando se selecciona "Nuevo" o se hace clic en el botón
706 def abrir_ventana_nueva():
707     global Ventana_Nuevo, checkbutton_1, checkbutton_2, checkbutton_3, checkbutton_4 ,
708     gif_player, combobox, label_prueba, spinbox, checkbutton_5, combo
709     global subventana_abierta
710     global entry_nombre_usuario, entry_ruta_archivo, entry_nombre_modelo,
711     text_detalle_modelo
712     global combo_concreto, entry_descripcion_concreto, combo_acero,
713     entry_descripcion_acero
714
715     # Verificar si la subventana ya está abierta
716     if Ventana_Nuevo is None or not Ventana_Nuevo.winfo_exists():
717         # Crear una nueva subventana (Toplevel)
718         Ventana_Nuevo = Toplevel(ventana)
719         Ventana_Nuevo.title("Nuevo Modelo")
720         Ventana_Nuevo.geometry("800x470") # Tamaño de la subventana
721
722         # Cargar datos para establecer el título
723         datos_cargados = cargar_datos()
724         nombre_modelo = datos_cargados.get("nombre_modelo", "")
725         Ventana_Nuevo.title(f"Nuevo Modelo - {nombre_modelo}" if nombre_modelo
726         else "Nuevo Modelo")
727
728         # Establecer un tamaño fijo y prevenir redimensionar
729         Ventana_Nuevo.resizable(False, False)
730
731         # Asegurarse de que la subventana esté siempre por encima de la ventana
732         principal
733         Ventana_Nuevo.attributes('-topmost', True)
734
735         # Crear LabelFrame en la nueva ventana
736         label_nuevo = tk.LabelFrame(Ventana_Nuevo, text="Configuraciones previas")
737         label_nuevo.place(x=10, y=10, width=200, height=400)
738
739         # Crear el Treeview dentro del LabelFrame
740         treeview = ttk.Treeview(label_nuevo)
741         treeview.place(x=10, y=10, width=170, height=300)
742
743         # Función para redimensionar imágenes
744         def cargar_icono(ruta, tamaño):
745             imagen = Image.open(ruta) # Abrir la imagen
746             imagen = imagen.resize(tamaño, Image.ANTIALIAS) # Redimensionar
747             return ImageTk.PhotoImage(imagen)
748
749         # Cargar las imágenes para los íconos
750         icon_general = tk.PhotoImage(file="Iconos_proyecto/boton_circular.png")
751         icon_unidades = tk.PhotoImage(file="Iconos_proyecto/boton_circular.png")
752         icon_materiales = tk.PhotoImage(file="Iconos_proyecto/boton_circular.png")
753         icon_modelo = tk.PhotoImage(file="Iconos_proyecto/boton_triangulo.png")
754         icono_concreto_acero =
755         tk.PhotoImage(file="Iconos_proyecto/Concreto_Acero.png")
```

```

752
753     # Mantener las imágenes en una variable para evitar que se eliminen por el
recolector de basura
754     treeview.icons = {
755         'general': icon_general,
756         'unidades': icon_unidades,
757         'materiales': icon_materiales,
758         'nuevo modelo': icon_modelo
759     }
760
761     # Agregar elementos al Treeview con jerarquía
762     # Opción principal
763     opcion1 = treeview.insert('', 'end', 'opcion1',
text='General', image=icon_general)
764     opcion2 = treeview.insert('', 'end', 'opcion2', text='Unidades /
Esfuerzos', image=icon_unidades)
765     opcion3 = treeview.insert('', 'end', 'opcion3',
text='Materiales', image=icon_materiales)
766
767     # Sub-elementos dentro de la Opción 1
768     treeview.insert(opcion1, 'end', 'subopcion1_1', text='Nuevo
Modelo', image=icon_modelo)
769     treeview.insert(opcion1, 'end', 'subopcion1_2', text='Estilo de
Modelo', image=icon_modelo)
770
771     # Sub-elementos dentro de la Opción 2
772     treeview.insert(opcion2, 'end', 'subopcion2_1',
text='Esfuerzos', image=icon_modelo)
773     # Sub-elementos dentro de la Opción 3
774     subopcion3_1 = treeview.insert(opcion3, 'end', 'subopcion3_1',
text="Concreto / Acero", image=icon_modelo)
775
776     # Seleccionar la opción "General" (sin expandir)
777     treeview.selection_set(opcion1)
778
779
780     # Función para bloquear la interacción con "opcion3" pero dejar ver el
subnodo
781     def bloquear_opcion3(event):
782         seleccion = treeview.identify_row(event.y) # Identificar qué fila se
clicó
783
784         if seleccion == "opcion3": # Si el clic es sobre "Materiales"
785             treeview.item(opcion3, open=True) # Expande el nodo de
"Materiales"
786             treeview.selection_set(subopcion3_1) # Forza la selección del
subelemento
787             treeview.focus(subopcion3_1) # Mueve el foco al subelemento
788             return "break" # Detiene la acción predeterminada de Treeview
(evita que se seleccione "Opción 3")
789
790     # Vincular un evento de clic para bloquear el acceso a "opcion3"
791     treeview.bind("<Button-1>", bloquear_opcion3)
792
793     # Inicializar la variable que indica si la subventana está abierta
794     subventana_abierta = False
795
796
797     # Función que abre la subventana si no está abierta
798     def abrir_subventana():

```

```

799     global subventana_abierta, boton_agregar
800     if not subventana_abierta: # Solo abre la subventana si no está
abierta
801         subventana_abierta = True # Marcar como abierta
802         subventana = Toplevel() # Crear la subventana
803         subventana.title("Configuración del material") # Título de la
subventana
804         subventana.geometry("394x280") # Tamaño de la subventana
805
806         # Hacer que la subventana esté siempre en el primer plano
807         subventana.attributes('-topmost', True)
808
809         # Evitar que la subventana sea redimensionable
810         subventana.resizable(False, False)
811
812         # Crear contenido dentro de la subventana
813         label_frame_concreto = tk.LabelFrame(subventana, text="Concreto
[f'c]")
814         label_frame_concreto.place(x=10,y=10,width=375,height=100)
815         label_materiales_concreto =
tk.Label(label_frame_concreto, text="Materiales:")
816         label_materiales_concreto.place(x=40,y=15)
817         label_materiales_descripcion_C =
tk.Label(label_frame_concreto, text="Descripción:")
818         label_materiales_descripcion_C.place(x=40,y=45)
819         entry_materiales_C = tk.Entry(label_frame_concreto)
820         entry_materiales_C.place(x=140,y=45,width=143)
821         label_frame_acero = tk.LabelFrame(subventana, text="Acero [fy]")
822         label_frame_acero.place(x=10,y=120,width=375,height=100)
823         label_materiales_acero =
tk.Label(label_frame_acero, text="Materiales:")
824         label_materiales_acero.place(x=40,y=15)
825         label_materiales_descripcion_A =
tk.Label(label_frame_acero, text="Descripción:")
826         label_materiales_descripcion_A.place(x=40,y=45)
827         entry_materiales_A = tk.Entry(label_frame_acero)
828         entry_materiales_A.place(x=140,y=45,width=143)
829         boton_agregar = ttk.Button(subventana, text="Agregar",
command=lambda: agregar_datos(entry_materiales_C, entry_materiales_A,
combo_materiales_fc, combo_materiales_fy))
830         boton_agregar.place(x=140,y=235,width=100)
831
832         # Guardamos la referencia de la subventana
833         abrir_subventana.subventana = subventana
834
835         # Crear un Combobox para f'c
836         combo_materiales_fc = ttk.Combobox(label_frame_concreto, values=[
837             "f'c = 17.5 MPa",
838             "f'c = 21 MPa",
839             "f'c = 24.5 MPa",
840             "f'c = 28 MPa",
841             "f'c = 35 MPa",
842             "f'c = 42 MPa"
843         ], state="readonly") # `readonly` para evitar edición manual
844         combo_materiales_fc.set("Concreto") # Valor por defecto
845         combo_materiales_fc.place(x=140,y=15)
846
847         # Función que actualiza el Entry de descripción de concreto según la
selección en el ComboBox
848         def actualizar_descripcion_concreto():

```

```

849     seleccion_fc = combo_materiales_fc.get()
850     if seleccion_fc == "f'c = 17.5 MPa":
851         entry_materiales_C.delete(0, tk.END) # Eliminar texto
actual
852         entry_materiales_C.insert(0, "Concreto 2500 Psi")
853     elif seleccion_fc == "f'c = 21 MPa":
854         entry_materiales_C.delete(0, tk.END)
855         entry_materiales_C.insert(0, "Concreto 3000 Psi")
856     elif seleccion_fc == "f'c = 24.5 MPa":
857         entry_materiales_C.delete(0, tk.END)
858         entry_materiales_C.insert(0, "Concreto 3500 Psi")
859     elif seleccion_fc == "f'c = 28 MPa":
860         entry_materiales_C.delete(0, tk.END)
861         entry_materiales_C.insert(0, "Concreto 4000 Psi")
862     elif seleccion_fc == "f'c = 35 MPa":
863         entry_materiales_C.delete(0, tk.END)
864         entry_materiales_C.insert(0, "Concreto 5000 Psi")
865     elif seleccion_fc == "f'c = 42 MPa":
866         entry_materiales_C.delete(0, tk.END)
867         entry_materiales_C.insert(0, "Concreto 6000 Psi")
868
869     # Vincular el evento de cambio en el Combobox de f'c
870     combo_materiales_fc.bind("<<ComboboxSelected>>", lambda event:
actualizar_descripcion_concreto())
871
872     # Crear un Combobox para fy
873     combo_materiales_fy = ttk.Combobox(label_frame_acero, values=[
874         "fy = 240 MPa",
875         "fy = 420 MPa"
876     ], state="readonly") # `readonly` para evitar edición manual
877     combo_materiales_fy.set("Acero") # Valor por defecto
878     combo_materiales_fy.place(x=140,y=15)
879
880     # Función que actualiza el Entry de descripción de acero según la
selección en el ComboBox
881     def actualizar_descripcion_acero():
882         seleccion_fy = combo_materiales_fy.get()
883         if seleccion_fy == "fy = 240 MPa":
884             entry_materiales_A.delete(0, tk.END) # Eliminar texto
actual
885             entry_materiales_A.insert(0, "Resistencia media")
886         elif seleccion_fy == "fy = 420 MPa":
887             entry_materiales_A.delete(0, tk.END)
888             entry_materiales_A.insert(0, "Resistencia alta")
889
890     # Vincular el evento de cambio en el Combobox de fy
891     combo_materiales_fy.bind("<<ComboboxSelected>>", lambda event:
actualizar_descripcion_acero())
892
893     # Función que se llama al hacer clic en el botón "Agregar"
894     def agregar_datos(entry_materiales_C, entry_materiales_A,
combo_materiales_fc, combo_materiales_fy):
895         # Copiar todos los valores de los ComboBox en los respectivos ComboBox de
la ventana principal
896         valores_fc = combo_materiales_fc['values']
897         valores_fy = combo_materiales_fy['values']
898
899         # Actualizar el ComboBox de la ventana principal con los valores
900         combo_concreto['values'] = valores_fc
901         combo_acero['values'] = valores_fy

```

```

902
903     # Si es necesario, se puede seleccionar el primer valor por defecto
904     combo_concreto.set(valores_fc[0])
905     combo_acero.set(valores_fy[0])
906
907     # Cerrar la subventana después de agregar los datos
908     subventana.destroy()
909
910     # Vincular el evento de cierre de la subventana
911     subventana.protocol("WM_DELETE_WINDOW", lambda:
cerrar_subventana(subventana))
912
913     def actualizar_descripcion_principal():
914         descripcion_concreto = {
915             "f'c = 17.5 MPa": "Concreto 2500 Psi",
916             "f'c = 21 MPa": "Concreto 3000 Psi",
917             "f'c = 24.5 MPa": "Concreto 3500 Psi",
918             "f'c = 28 MPa": "Concreto 4000 Psi",
919             "f'c = 35 MPa": "Concreto 5000 Psi",
920             "f'c = 42 MPa": "Concreto 6000 Psi"
921         }
922         entry_descripcion_concreto.delete(0, tk.END)
923         entry_descripcion_concreto.insert(0,
descripcion_concreto.get(combo_concreto.get(), ""))
924
925         combo_concreto.bind("<<ComboboxSelected>>", lambda event:
actualizar_descripcion_principal())
926
927     def actualizar_descripcion_acero_principal():
928         descripcion_acero = {
929             "fy = 240 MPa": "Resistencia media",
930             "fy = 420 MPa": "Resistencia alta"
931         }
932         entry_descripcion_acero.delete(0, tk.END)
933         entry_descripcion_acero.insert(0,
descripcion_acero.get(combo_acero.get(), ""))
934
935     # Vincular evento de cambio en combo_acero
936     combo_acero.bind("<<ComboboxSelected>>", lambda event:
actualizar_descripcion_acero_principal())
937
938
939     # Función para cerrar la subventana y restablecer la variable global
940     def cerrar_subventana(subventana):
941         global subventana_abierta
942         subventana.destroy() # Cierra la subventana
943         subventana_abierta = False # Restablece el estado para permitir
abrir la nuevamente
944
945
946     # Función que actualizará la información en el lado derecho
947     def mostrar_informacion(event):
948         global combos, subventana_abierta
949         # Obtener el ítem seleccionado
950         item = treeview.selection()
951         if item:
952             # Aquí puedes ajustar la información según la selección
953             info = f"{treeview.item(item)['text']}"
954             label_info.config(text=info)
955

```

```

956
957
958
959 # Mostrar el campo "Nombre de Usuario" si la opción seleccionada
es 'General'
960 if treeview.item(item)['text'] == 'General':
961     label_nombre_usuario.place(x=40, y=50)
962     entry_nombre_usuario.place(x=170, y=50, width=168)
963     label_fecha_entrada.place(x=350,y=50)
964     entry_fecha_entrada.place(x=400,y=50,width=70)
965     labelFrame_archivo_abrir.place(x=40,y=100,width=480,height=80)
966     label_ruta_archivo.place(x=10,y=10)
967     entry_ruta_archivo.place(x=110,y=10,width=300)
968     labelFrame_archivo_información.place(x=40,y=190, width=480,
height=80)
969     label_ruta_usuario.place(x=10,y=10)
970     entry_ruta_usuario.place(x=110,y=10,width=300)
971     labelFrame_opciones.place(x=40,y=280, width=480, height=80)
972     checkbox_1.place(x=10,y=10)
973
974
975 # Ocultar otros widgets si no es la opción correcta
976 label_nombre_modelo.place_forget()
977 label_descripcion_modelo.place_forget()
978 entry_nombre_modelo.place_forget()
979 text_detalle_modelo.place_forget()
980 labelFrame_complementos.place_forget()
981 checkbox_2.place_forget()
982 checkbox_3.place_forget()
983 checkbox_4.place_forget()
984 labelFrame_estilo.place_forget()
985 canvas_viga.place_forget()
986 labelFrame_contenido.place_forget()
987 combobox.place_forget()
988 label_tipo_viga.place_forget()
989 label_mensaje_ayuda.place_forget()
990 label_contenido_1.place_forget()
991 label_mensaje_ayuda_1.place_forget()
992 boton_aplicar.place_forget()
993 label_prueba.place_forget()
994 label_formato.place_forget()
995 formato_decimal.place_forget()
996 slider_1.place_forget()
997 label_mensaje_formato.place_forget()
998 label_formato_unidades.place_forget()
999 label_unidades_1.place_forget()
1000 label_unidades_2.place_forget()
1001 label_unidades_3.place_forget()
1002 label_unidades_4.place_forget()
1003 combo1.place_forget()
1004 combo2.place_forget()
1005 combo3.place_forget()
1006 combo4.place_forget()
1007 checkbox_5.place_forget()
1008 labelFrame_Esfuerzos.place_forget()
1009 label_longitud.place_forget()
1010 label_fuerza.place_forget()
1011 label_Momento.place_forget()
1012 label_Tension.place_forget()
1013 longitud_1.place_forget()

```

```

1014         Fuerza_1.place_forget()
1015         entry_momento.place_forget()
1016         entry_tension.place_forget()
1017         checkbutton_6.place_forget()
1018         labelFrame_Nota.place_forget()
1019         label_mensaje_nota.place_forget()
1020         labelFrame_Materiales.place_forget()
1021         label_concreto.place_forget()
1022         label_descripcion_concreto.place_forget()
1023         label_acero.place_forget()
1024         label_descripcion_acero.place_forget()
1025         combo_concreto.place_forget()
1026         entry_descripcion_concreto.place_forget()
1027         combo_acero.place_forget()
1028         entry_descripcion_acero.place_forget()
1029         labelFrame_boton.place_forget()
1030         label_configuracion.place_forget()
1031
1032
1033
1034         # Mostrar el campo "Nombre del modelo" si la opción
seleccionada es 'Nuevo Modelo'
1035         elif treeview.item(item)['text'] == 'Nuevo Modelo':
1036             label_nombre_modelo.place(x=40, y=50)
1037             label_descripcion_modelo.place(x=40, y=100)
1038             entry_nombre_modelo.place(x=170, y=50, width=320)
1039             text_detalle_modelo.place(x=40, y=140, width=450, height=100)
1040             labelFrame_complementos.place(x=40, y=250, width=450, height=100)
1041             checkbutton_2.place(x=10, y=10)
1042             checkbutton_3.place(x=230, y=10)
1043             checkbutton_4.place(x=10, y=50)
1044
1045         # Ocultar otros widgets si no es la opción correcta
1046         label_nombre_usuario.place_forget()
1047         entry_nombre_usuario.place_forget()
1048         label_fecha_entrada.place_forget()
1049         entry_fecha_entrada.place_forget()
1050         labelFrame_archivo_abrir.place_forget()
1051         label_ruta_archivo.place_forget()
1052         entry_ruta_archivo.place_forget()
1053         labelFrame_archivo_información.place_forget()
1054         label_ruta_usuario.place_forget()
1055         entry_ruta_usuario.place_forget()
1056         labelFrame_opciones.place_forget()
1057         checkbutton_1.place_forget()
1058         labelFrame_estilo.place_forget()
1059         canvas_viga.place_forget()
1060         labelFrame_contenido.place_forget()
1061         combobox.place_forget()
1062         label_tipo_viga.place_forget()
1063         label_mensaje_ayuda.place_forget()
1064         label_contenido_1.place_forget()
1065         label_mensaje_ayuda_1.place_forget()
1066         boton_aplicar.place_forget()
1067         label_prueba.place_forget()
1068         label_formato.place_forget()
1069         formato_decimal.place_forget()
1070         slider_1.place_forget()
1071         label_mensaje_formato.place_forget()
1072         label_formato_unidades.place_forget()

```

```
1073 label_unidades_1.place_forget()
1074 label_unidades_2.place_forget()
1075 label_unidades_3.place_forget()
1076 label_unidades_4.place_forget()
1077 combo1.place_forget()
1078 combo2.place_forget()
1079 combo3.place_forget()
1080 combo4.place_forget()
1081 checkbutton_5.place_forget()
1082 labelFrame_Esfuerzos.place_forget()
1083 label_longitud.place_forget()
1084 label_fuerza.place_forget()
1085 label_Momento.place_forget()
1086 label_Tension.place_forget()
1087 longitud_1.place_forget()
1088 Fuerza_1.place_forget()
1089 entry_momento.place_forget()
1090 entry_tension.place_forget()
1091 checkbutton_6.place_forget()
1092 labelFrame_Nota.place_forget()
1093 label_mensaje_nota.place_forget()
1094 labelFrame_Materiales.place_forget()
1095 label_concreto.place_forget()
1096 label_descripcion_concreto.place_forget()
1097 label_acero.place_forget()
1098 label_descripcion_acero.place_forget()
1099 combo_concreto.place_forget()
1100 entry_descripcion_concreto.place_forget()
1101 combo_acero.place_forget()
1102 entry_descripcion_acero.place_forget()
1103 labelFrame_boton.place_forget()
1104 label_configuracion.place_forget()
1105
1106
1107
1108
1109
1110 # Mostrar el LabelFrame vacío para 'Estilo de Modelo'
1111 elif treeview.item(item)['text'] == 'Estilo de Modelo':
1112     labelFrame_estilo.place(x=30, y=50, width=200, height=200)
1113     canvas_viga.place(x=2,y=2,width=190,height=170)
1114     labelFrame_contenido.place(x=250,y=50,width=280, height=200)
1115     combobox.place(x=100,y=20)
1116     label_tipo_viga.place(x=20,y=20)
1117     label_mensaje_ayuda.place(x=20,y=60,width=230,height=100)
1118     label_contenido_1.place(x=30,y=260,width=501,height=100)
1119     label_mensaje_ayuda_1.place(x=10,y=10)
1120     boton_aplicar.place(x=200,y=40,width=100)
1121     #label_prueba.place(x=50,y=40,width=150)
1122
1123 # Ocultar otros widgets si no es la opción correcta
1124 label_nombre_modelo.place_forget()
1125 label_descripcion_modelo.place_forget()
1126 entry_nombre_modelo.place_forget()
1127 text_detalle_modelo.place_forget()
1128 labelFrame_complementos.place_forget()
1129 checkbutton_2.place_forget()
1130 checkbutton_3.place_forget()
1131 checkbutton_4.place_forget()
1132 label_nombre_usuario.place_forget()
```

```
1133     entry_nombre_usuario.place_forget()
1134     label_fecha_entrada.place_forget()
1135     entry_fecha_entrada.place_forget()
1136     labelFrame_archivo_abrir.place_forget()
1137     label_ruta_archivo.place_forget()
1138     entry_ruta_archivo.place_forget()
1139     labelFrame_archivo_información.place_forget()
1140     label_ruta_usuario.place_forget()
1141     entry_ruta_usuario.place_forget()
1142     labelFrame_opciones.place_forget()
1143     checkbutton_1.place_forget()
1144     label_formato.place_forget()
1145     formato_decimal.place_forget()
1146     slider_1.place_forget()
1147     label_mensaje_formato.place_forget()
1148     label_formato_unidades.place_forget()
1149     label_unidades_1.place_forget()
1150     label_unidades_2.place_forget()
1151     label_unidades_3.place_forget()
1152     label_unidades_4.place_forget()
1153     combo1.place_forget()
1154     combo2.place_forget()
1155     combo3.place_forget()
1156     combo4.place_forget()
1157     checkbutton_5.place_forget()
1158     labelFrame_Esfuerzos.place_forget()
1159     label_longitud.place_forget()
1160     label_fuerza.place_forget()
1161     label_Momento.place_forget()
1162     label_Tension.place_forget()
1163     longitud_1.place_forget()
1164     Fuerza_1.place_forget()
1165     entry_momento.place_forget()
1166     entry_tension.place_forget()
1167     checkbutton_6.place_forget()
1168     labelFrame_Nota.place_forget()
1169     label_mensaje_nota.place_forget()
1170     labelFrame_Materiales.place_forget()
1171     label_concreto.place_forget()
1172     label_descripcion_concreto.place_forget()
1173     label_acero.place_forget()
1174     label_descripcion_acero.place_forget()
1175     combo_concreto.place_forget()
1176     entry_descripcion_concreto.place_forget()
1177     combo_acero.place_forget()
1178     entry_descripcion_acero.place_forget()
1179     labelFrame_boton.place_forget()
1180     label_configuracion.place_forget()
1181
1182
1183
1184     # Mostrar el contenido para 'Unidades / Fuerzas'
1185     elif treeview.item(item)['text'] == 'Unidades / Esfuerzos':
1186         label_formato.place(x=50,y=60,width=460,height=100)
1187         formato_decimal.place(x=20,y=30)
1188         slider_1.place(x=250,y=10)
1189         label_mensaje_formato.place(x=20,y=60)
1190         label_formato_unidades.place(x=50,y=180,width=460,height=180)
1191         label_unidades_1.place(x=20,y=10)
1192         label_unidades_2.place(x=20,y=40)
```

```
1193 label_unidades_3.place(x=20,y=70)
1194 label_unidades_4.place(x=20,y=100)
1195 combo1.place(x=220,y=10)
1196 combo2.place(x=220,y=40)
1197 combo3.place(x=220,y=70)
1198 combo4.place(x=220,y=100)
1199 checkbox_5.place(x=20,y=135)
1200
1201
1202
1203 # Ocultar otros widgets si no es la opción correcta
1204 label_nombre_modelo.place_forget()
1205 label_descripcion_modelo.place_forget()
1206 entry_nombre_modelo.place_forget()
1207 text_detalle_modelo.place_forget()
1208 labelFrame_complementos.place_forget()
1209 checkbox_2.place_forget()
1210 checkbox_3.place_forget()
1211 checkbox_4.place_forget()
1212 label_nombre_usuario.place_forget()
1213 entry_nombre_usuario.place_forget()
1214 label_fecha_entrada.place_forget()
1215 entry_fecha_entrada.place_forget()
1216 labelFrame_archivo_abrir.place_forget()
1217 label_ruta_archivo.place_forget()
1218 entry_ruta_archivo.place_forget()
1219 labelFrame_archivo_información.place_forget()
1220 label_ruta_usuario.place_forget()
1221 entry_ruta_usuario.place_forget()
1222 labelFrame_opciones.place_forget()
1223 checkbox_1.place_forget()
1224 labelFrame_estilo.place_forget()
1225 canvas_viga.place_forget()
1226 labelFrame_contenido.place_forget()
1227 combobox.place_forget()
1228 label_tipo_viga.place_forget()
1229 label_mensaje_ayuda.place_forget()
1230 label_contenido_1.place_forget()
1231 label_mensaje_ayuda_1.place_forget()
1232 boton_aplicar.place_forget()
1233 label_prueba.place_forget()
1234 labelFrame_Esfuerzos.place_forget()
1235 label_longitud.place_forget()
1236 label_fuerza.place_forget()
1237 label_Momento.place_forget()
1238 label_Tension.place_forget()
1239 longitud_1.place_forget()
1240 Fuerza_1.place_forget()
1241 entry_momento.place_forget()
1242 entry_tension.place_forget()
1243 checkbox_6.place_forget()
1244 labelFrame_Nota.place_forget()
1245 label_mensaje_nota.place_forget()
1246 labelFrame_Materiales.place_forget()
1247 label_concreto.place_forget()
1248 label_descripcion_concreto.place_forget()
1249 label_acero.place_forget()
1250 label_descripcion_acero.place_forget()
1251 combo_concreto.place_forget()
1252 entry_descripcion_concreto.place_forget()
```

```
1253         combo_acero.place_forget()
1254         entry_descripcion_acero.place_forget()
1255         labelFrame_boton.place_forget()
1256         label_configuracion.place_forget()
1257
1258
1259
1260
1261         # Mostrar el contenido para 'Unidades / Fuerzas'
1262         elif treeview.item(item)['text'] == 'Esfuerzos':
1263             labelFrame_Esfuerzos.place(x=50,y=60,width=460,height=250)
1264             label_longitud.place(x=20,y=30)
1265             label_fuerza.place(x=20,y=70)
1266             label_Momento.place(x=20,y=110)
1267             label_Tension.place(x=20,y=150)
1268             longitud_1.place(x=200,y=30)
1269             Fuerza_1.place(x=200,y=70)
1270             entry_momento.place(x=200,y=110,width=143)
1271             entry_tension.place(x=200,y=150,width=143)
1272             checkbutton_6.place(x=30,y=200)
1273             labelFrame_Nota.place(x=50,y=320,width=460,height=50)
1274             label_mensaje_nota.place(x=10,y=10)
1275
1276
1277
1278
1279
1280         # Ocultar otros widgets si no es la opción correcta
1281         label_nombre_modelo.place_forget()
1282         label_descripcion_modelo.place_forget()
1283         entry_nombre_modelo.place_forget()
1284         text_detalle_modelo.place_forget()
1285         label_nombre_usuario.place_forget()
1286         entry_nombre_usuario.place_forget()
1287         label_fecha_entrada.place_forget()
1288         entry_fecha_entrada.place_forget()
1289         labelFrame_archivo_abrir.place_forget()
1290         label_ruta_archivo.place_forget()
1291         entry_ruta_archivo.place_forget()
1292         labelFrame_archivo_información.place_forget()
1293         label_ruta_usuario.place_forget()
1294         entry_ruta_usuario.place_forget()
1295         labelFrame_opciones.place_forget()
1296         checkbutton_1.place_forget()
1297         labelFrame_complementos.place_forget()
1298         checkbutton_2.place_forget()
1299         checkbutton_3.place_forget()
1300         checkbutton_4.place_forget()
1301         labelFrame_estilo.place_forget()
1302         canvas_viga.place_forget()
1303         labelFrame_contenido.place_forget()
1304         combobox.place_forget()
1305         label_tipo_viga.place_forget()
1306         label_mensaje_ayuda.place_forget()
1307         label_contenido_1.place_forget()
1308         label_mensaje_ayuda_1.place_forget()
1309         boton_aplicar.place_forget()
1310         label_prueba.place_forget()
1311         label_formato.place_forget()
1312         formato_decimal.place_forget()
```

```

1313 slider_1.place_forget()
1314 label_mensaje_formato.place_forget()
1315 label_formato_unidades.place_forget()
1316 label_unidades_1.place_forget()
1317 label_unidades_2.place_forget()
1318 label_unidades_3.place_forget()
1319 label_unidades_4.place_forget()
1320 combo1.place_forget()
1321 combo2.place_forget()
1322 combo3.place_forget()
1323 combo4.place_forget()
1324 checkbutton_5.place_forget()
1325 labelFrame_Materiales.place_forget()
1326 label_concreto.place_forget()
1327 label_descripcion_concreto.place_forget()
1328 label_acero.place_forget()
1329 label_descripcion_acero.place_forget()
1330 combo_concreto.place_forget()
1331 entry_descripcion_concreto.place_forget()
1332 combo_acero.place_forget()
1333 entry_descripcion_acero.place_forget()
1334 labelFrame_boton.place_forget()
1335 label_configuracion.place_forget()
1336
1337
1338 # Mostrar el contenido para 'Unidades / Fuerzas'
1339 if treeview.item(item)['text'] == "Concreto / Acero":
1340     labelFrame_Materiales.place(x=50,y=60,width=460,height=260)
1341     label_concreto.place(x=80,y=30)
1342     label_descripcion_concreto.place(x=80,y=80)
1343     label_acero.place(x=80,y=130)
1344     label_descripcion_acero.place(x=80,y=180)
1345     combo_concreto.place(x=200,y=30)
1346     entry_descripcion_concreto.place(x=200,y=80,width=142)
1347     combo_acero.place(x=200,y=130)
1348     entry_descripcion_acero.place(x=200,y=180,width=142)
1349     labelFrame_boton.place(x=50,y=30,width=460,height=30)
1350     boton_concreto_acero = tk.Button(labelFrame_Materiales,
image=icono_concreto_acero, compound="top",command=abrir_subventana)
1351     boton_concreto_acero.place(x=420,y=1)
1352     label_configuracion.place(x=1,y=5)
1353
1354
1355 # Ocultar todos los widgets si se selecciona otra opción
1356 label_nombre_modelo.place_forget()
1357 label_descripcion_modelo.place_forget()
1358 entry_nombre_modelo.place_forget()
1359 text_detalles_modelo.place_forget()
1360 label_nombre_usuario.place_forget()
1361 entry_nombre_usuario.place_forget()
1362 label_fecha_entrada.place_forget()
1363 entry_fecha_entrada.place_forget()
1364 labelFrame_archivo_abrir.place_forget()
1365 label_ruta_archivo.place_forget()
1366 entry_ruta_archivo.place_forget()
1367 labelFrame_archivo_información.place_forget()
1368 label_ruta_usuario.place_forget()
1369 entry_ruta_usuario.place_forget()
1370 labelFrame_opciones.place_forget()
1371 checkbutton_1.place_forget()

```

```
1372 labelFrame_complementos.place_forget()
1373 checkbutton_2.place_forget()
1374 checkbutton_3.place_forget()
1375 checkbutton_4.place_forget()
1376 labelFrame_estilo.place_forget()
1377 canvas_viga.place_forget()
1378 labelFrame_contenido.place_forget()
1379 combobox.place_forget()
1380 label_tipo_viga.place_forget()
1381 label_mensaje_ayuda.place_forget()
1382 label_contenido_1.place_forget()
1383 label_mensaje_ayuda_1.place_forget()
1384 boton_aplicar.place_forget()
1385 label_prueba.place_forget()
1386 label_formato.place_forget()
1387 formato_decimal.place_forget()
1388 slider_1.place_forget()
1389 label_mensaje_formato.place_forget()
1390 label_formato_unidades.place_forget()
1391 label_unidades_1.place_forget()
1392 label_unidades_2.place_forget()
1393 label_unidades_3.place_forget()
1394 label_unidades_4.place_forget()
1395 combo1.place_forget()
1396 combo2.place_forget()
1397 combo3.place_forget()
1398 combo4.place_forget()
1399 checkbutton_5.place_forget()
1400 labelFrame_Esfuerzos.place_forget()
1401 label_longitud.place_forget()
1402 label_fuerza.place_forget()
1403 label_Momento.place_forget()
1404 label_Tension.place_forget()
1405 longitud_1.place_forget()
1406 Fuerza_1.place_forget()
1407 entry_momento.place_forget()
1408 entry_tension.place_forget()
1409 checkbutton_6.place_forget()
1410 labelFrame_Nota.place_forget()
1411 label_mensaje_nota.place_forget()
1412
1413
1414
1415
1416 # Vincular el evento de selección para mostrar información
1417 treeview.bind('<<TreeviewSelect>>', mostrar_informacion)
1418
1419
1420 # Crear el frame para mostrar la información (lado derecho)
1421 frame_derecha = tk.Frame(Ventana_Nuevo, bd=2, relief="raise")
1422 frame_derecha.place(x=220, y=17, width=560, height=393)
1423
1424 # Crear una etiqueta para mostrar la información
1425 label_info = tk.LabelFrame(frame_derecha, text="")
1426 label_info.place(x=12, y=12, width=535, height=366)
1427
1428 # Agregar un Label y Entry para "Nuevo modelo" y widgets
1429 label_nombre_modelo = tk.Label(frame_derecha, text="Nombre del modelo:")
1430 label_descripcion_modelo = tk.Label(frame_derecha, text="Descripción:")
1431 entry_nombre_modelo = tk.Entry(frame_derecha)
```

```
1432     text_detalle_modelo = tk.Text(frame_derecha)
1433     labelFrame_complementos = tk.LabelFrame(frame_derecha, text="Complementos
adicionales")
1434
1435     # Agregar un Label y Entru para "General" y widgets
1436     label_nombre_usuario = tk.Label(frame_derecha, text="Nombre de usuario:")
1437     entry_nombre_usuario = tk.Entry(frame_derecha)
1438     label_fecha_entrada = tk.Label(frame_derecha, text="Fecha:")
1439     entry_fecha_entrada = tk.Entry(frame_derecha)
1440     labelFrame_archivo_abrir = tk.LabelFrame(frame_derecha, text="Guardar \
Archivo")
1441     label_ruta_archivo = tk.Label(labelFrame_archivo_abrir, text="Ruta del
archivo:")
1442     entry_ruta_archivo = tk.Entry(labelFrame_archivo_abrir)
1443     labelFrame_archivo_información = tk.LabelFrame(frame_derecha,
text="Guardar \ Información \ Usuario")
1444     label_ruta_usuario = tk.Label(labelFrame_archivo_información, text="Ruta
del usuario:")
1445     entry_ruta_usuario = tk.Entry(labelFrame_archivo_información)
1446     labelFrame_opciones = tk.LabelFrame(frame_derecha, text="Opciones de
autoguardado")
1447
1448     # Agregar un Label y Entru para "Estilo de Modelo" y widgets
1449     labelFrame_estilo = tk.LabelFrame(frame_derecha, text="Plantilla")
1450     canvas_viga = tk.Canvas(labelFrame_estilo)
1451     labelFrame_contenido = tk.LabelFrame(frame_derecha, text="Selección de
modelo")
1452     label_tipo_viga = tk.Label(labelFrame_contenido, text="Tipo Viga:")
1453     label_mensaje_ayuda = tk.Label(labelFrame_contenido, text="", width=30,
height=5, bg="white", bd=2, relief="groove", anchor="w", justify="left",
wraplength=230)
1454     label_contenido_1 = tk.LabelFrame(frame_derecha, text="Visualización de
modelo")
1455     label_mensaje_ayuda_1 = tk.Label(label_contenido_1, text="Para ver la
visualización del modelo en la ventana principal, haz clic en el botón 'Aplicar'")
1456     label_prueba = tk.Label(label_contenido_1, text="")
1457
1458
1459     # Agregar un Label y Entru para "Unidades / Esfuerzos" y widgets
1460     label_formato = tk.LabelFrame(frame_derecha, text="")
1461     formato_decimal = tk.Label(label_formato, text="Valor numérico en formato
decimal:")
1462     label_mensaje_formato = tk.Label(label_formato, text="")
1463     label_formato_unidades = tk.LabelFrame(frame_derecha, text="")
1464     label_unidades_1 = tk.Label(label_formato_unidades, text="Dimensiones de la
viga:")
1465     label_unidades_2 = tk.Label(label_formato_unidades, text="Dimensiones de la
sección:")
1466     label_unidades_3 = tk.Label(label_formato_unidades, text="Diámetro barras
de refuerzo:")
1467     label_unidades_4 = tk.Label(label_formato_unidades, text="Área de sección
de refuerzo:")
1468
1469     # Agregar un Label y Entru para "Esfuerzos" y widgets
1470     labelFrame_Esfuerzos = tk.LabelFrame(frame_derecha, text="")
1471     label_longitud = tk.Label(labelFrame_Esfuerzos, text="Unidad de longitud:")
1472     label_fuerza = tk.Label(labelFrame_Esfuerzos, text="Unidad de Fuerza:")
1473     label_Momento = tk.Label(labelFrame_Esfuerzos, text="Unidad de Momento:")
1474     label_Tension = tk.Label(labelFrame_Esfuerzos, text="Unidad de Tensión:")
1475     entry_momento = tk.Entry(labelFrame_Esfuerzos)
```

```

1476     entry_tension = tk.Entry(labelFrame_Esfuerzos)
1477     labelFrame_Nota = tk.LabelFrame(frame_derecha,text="",bd=0, relief="flat")
1478     label_mensaje_nota = tk.Label(labelFrame_Nota,text="")
1479
1480     # Agregar un Label y Entru para "Concreto / Acero" y widgets
1481     labelFrame_Materiales = tk.LabelFrame(frame_derecha,text="Materiales")
1482     label_concreto = tk.Label(labelFrame_Materiales,text= "Concreto [f'c]:")
1483     label_descripcion_concreto =
tk.Label(labelFrame_Materiales,text="Descripción:")
1484     label_acero = tk.Label(labelFrame_Materiales,text= "Acero [fy]:")
1485     label_descripcion_acero =
tk.Label(labelFrame_Materiales,text="Descripción:")
1486     combo_concreto= ttk.Combobox(labelFrame_Materiales)
1487     entry_descripcion_concreto= tk.Entry(labelFrame_Materiales)
1488     combo_acero = ttk.Combobox(labelFrame_Materiales)
1489     entry_descripcion_acero= tk.Entry(labelFrame_Materiales)
1490     labelFrame_boton = tk.LabelFrame(frame_derecha,text="",bd=0,
relief="flat")
1491     label_configuracion = tk.Label(labelFrame_boton,text="Nota: Haz clic en el
ícono en la esquina superior derecha para cargar los materiales.")
1492
1493     # Cargar datos si existen y asignar valores al Combobox
1494     datos_cargados = cargar_datos()
1495     opciones_concreto = [
1496         "f'c = 17.5 MPa",
1497         "f'c = 21 MPa",
1498         "f'c = 24.5 MPa",
1499         "f'c = 28 MPa",
1500         "f'c = 35 MPa",
1501         "f'c = 42 MPa",
1502     ]
1503     opciones_acero = [
1504         "fy = 240 MPa",
1505         "fy = 420 MPa",
1506     ]
1507     combo_concreto["values"] = opciones_concreto
1508     combo_acero["values"] = opciones_acero
1509
1510     if datos_cargados:
1511         entry_nombre_usuario.insert(0, datos_cargados.get("nombre_usuario",
""))
1512         entry_nombre_modelo.insert(0, datos_cargados.get("nombre_modelo", ""))
1513         text_detalle_modelo.insert("1.0",
datos_cargados.get("detalles_modelo", ""))
1514         combo_concreto.set(datos_cargados.get("combo_concreto",
opciones_concreto[0]))
1515         combo_acero.set(datos_cargados.get("combo_acero", opciones_acero[0]))
1516
1517     else:
1518         Ventana_Nuevo.focus_set()
1519
1520
1521     # Botón "Aceptar" para guardar los datos
1522     boton_aceptar = ttk.Button(Ventana_Nuevo,
text="Aceptar",command=guardar_datos)
1523     boton_aceptar.place(x=350,y=430,width=100)
1524
1525
1526     # Lista con las opciones de unidades Longitud/Fuerza
1527

```

```

1528     Longitud= ["m"]
1529     Fuerza = ["kN"]
1530
1531     # Crear el ComboBox y agregar las opciones
1532     longitud_1 = ttk.Combobox(labelFrame_Esfuerzos, values=Longitud)
1533     longitud_1.set("Seleccionar")
1534     Fuerza_1 = ttk.Combobox(labelFrame_Esfuerzos, values=Fuerza)
1535     Fuerza_1.set("Seleccionar")
1536
1537
1538     # Función que actualiza los valores en los Entry cuando se seleccionan las
unidades
1539     def actualizar_unidades(event):
1540         # Obtener los valores seleccionados en los ComboBox
1541         unidad_fuerza = Fuerza_1.get()
1542         unidad_longitud = longitud_1.get()
1543
1544         # Comprobar si ambos ComboBox tienen opciones seleccionadas
1545         if unidad_fuerza != "Seleccionar" and unidad_longitud !=
"Seleccionar":
1546             # Actualizar el valor del momento y la tensión
1547             entry_momento.delete(0, tk.END)
1548             entry_tension.delete(0, tk.END)
1549
1550             entry_momento.insert(0, f"{unidad_fuerza}{unidad_longitud}") #
Ejemplo: kNm
1551             entry_tension.insert(0, f"{unidad_fuerza}{unidad_longitud}^2") #
Ejemplo: kNm²
1552
1553         # Vincular la función al evento de cambio de selección en los ComboBox
1554         Fuerza_1.bind("<<ComboboxSelected>>", actualizar_unidades)
1555         longitud_1.bind("<<ComboboxSelected>>", actualizar_unidades)
1556
1557         # Lista con las opciones de unidades
1558         unidades = ["metros", "centímetros", "milímetros"]
1559
1560         # Crear el ComboBox y agregar las opciones
1561         combo1 = ttk.Combobox(label_formato_unidades, values=unidades)
1562         combo1.set("Seleccionar") # Establecer el valor por defecto
1563
1564         combo2 = ttk.Combobox(label_formato_unidades, values=unidades)
1565         combo2.set("Seleccionar") # Establecer el valor por defecto
1566
1567         combo3 = ttk.Combobox(label_formato_unidades, values=unidades)
1568         combo3.set("Seleccionar") # Establecer el valor por defecto
1569
1570         combo4= ttk.Combobox(label_formato_unidades, values=unidades)
1571         combo4.set("Seleccionar") # Establecer el valor por defecto
1572
1573
1574         # Combobox dentro de labelFrame_contenido
1575         combobox = ttk.Combobox(labelFrame_contenido, values=["Viga apoyada", "Viga
empotrada "])
1576         combobox.set("Selecciona una opción")
1577         combobox.bind("<<ComboboxSelected>>", lambda event:
on_combobox_select(event, canvas_viga,label_mensaje_ayuda,label_prueba))
1578
1579         # Botón para visualizar modelo viga
1580         boton_aplicar= ttk.Button(label_contenido_1,text="Aplicar",command=lambda:
on_boton_aplicar_click(label_contenido_1, label_prueba))

```

```
1581
1582     def update_label(val):
1583         # Convertir el valor a entero y calcular cuántos ceros agregar
1584         num_ceros = int(val) # El valor del slider es un número entero que
indica los ceros
1585         formatted_value = "0." + "0" * num_ceros # Construir el número con
'num_ceros' ceros
1586         label_mensaje_formato.config(text=formatted_value) # Actualizar la
etiqueta con el valor formateado
1587
1588         # Crear el mensaje con el valor del slider y la recomendación
1589         mensaje_slider = f"Seleccionaste el valor en formato decimal con una
precisión de: {formatted_value}"
1590         label_mensaje_formato.config(text=mensaje_slider) # Actualizar el
mensaje en el label
1591
1592
1593         # Configurar el slider (de 0 a 4, con incremento de 1, orientado
horizontalmente)
1594         slider_1 = tk.Scale(label_formato, from_=0, to=4, resolution=1,
orient="horizontal", command=update_label)
1595         slider_1.set(0) # Establecer valor inicial en 0
1596
1597         # Crear widgets para "General"
1598         # Variable de control para el checkbox
1599         var = tk.IntVar()
1600         var1 = tk.IntVar()
1601         var2 = tk.IntVar()
1602         var3 = tk.IntVar()
1603         var4 = tk.IntVar()
1604         var5 = tk.IntVar()
1605
1606         # Función para cambiar el color del chulito cuando se marca o desmarca
1607         def cambiar_color(checkbutton, var):
1608             if var.get() == 1:
1609                 checkbutton.config(selectcolor="blue")
1610             else:
1611                 checkbutton.config(selectcolor="")
1612
1613
1614         # Función para cambiar el formato de los ComboBoxes cuando se activa el
Checkbutton
1615         def aplicar_formato_predefinido():
1616             if var4.get() == 1: # Si el Checkbutton está activado
1617                 combo1.set("metros") # Primer ComboBox -> "metros"
1618                 combo2.set("metros") # Segundo ComboBox -> "metros"
1619                 combo3.set("milímetros") # Tercer ComboBox -> "milímetros"
1620                 combo4.set("milímetros") # Cuarto ComboBox -> "milímetros"
1621             else:
1622                 # Si el Checkbutton está desactivado, los ComboBoxes se reinician a
"Seleccionar"
1623                 combo1.set("Seleccionar")
1624                 combo2.set("Seleccionar")
1625                 combo3.set("Seleccionar")
1626                 combo4.set("Seleccionar")
1627
1628         # Función para manejar el Checkbutton y seleccionar valores por defecto
1629         def cambiar_a_valores_por_defecto():
1630             if var5.get(): # Si el Checkbutton está marcado
1631                 # Establecer los valores predeterminados en los ComboBox
```

```
1632         Fuerza_1.set("kN")
1633         longitud_1.set("m")
1634
1635         # Actualizar los campos de entrada
1636         entry_momento.delete(0, tk.END)
1637         entry_tension.delete(0, tk.END)
1638         entry_momento.insert(0, "kNm") # Valor predeterminado para el
momento
1639         entry_tension.insert(0, "kNm²") # Valor predeterminado para la
tensión
1640
1641         # Mostrar un mensaje interactivo en el label
1642         label_mensaje_notas.config(text="¡Muy bien! Ahora continua con la
configuración de los materiales.")
1643
1644         else:
1645             # Si el Checkbutton está desmarcado, dejar las opciones en blanco o por
defecto
1646             Fuerza_1.set("Seleccionar")
1647             longitud_1.set("Seleccionar")
1648             entry_momento.delete(0, tk.END)
1649             entry_tension.delete(0, tk.END)
1650             entry_momento.insert(0, "Momento") # Valor predeterminado
(opcional)
1651             entry_tension.insert(0, "Tensión") # Valor predeterminado
(opcional)
1652
1653             # Crear el Checkbutton personalizado
1654             checkbutton_1 = tk.Checkbutton(
1655                 labelFrame_opciones,
1656                 text="Guardar información predeterminada",
1657                 variable=var,
1658                 onvalue=1,
1659                 offvalue=0,
1660                 indicatoron=True,
1661                 selectcolor="",
1662                 command=lambda: cambiar_color(checkbutton_1, var)
1663             )
1664
1665             # Crear el Checkbutton para "Generar documento Word"
1666             checkbutton_2 = tk.Checkbutton(
1667                 labelFrame_complementos,
1668                 text="Generar documento Word",
1669                 variable=var1,
1670                 onvalue=1,
1671                 offvalue=0,
1672                 indicatoron=True,
1673                 selectcolor="",
1674                 command=lambda: cambiar_color(checkbutton_2, var1)
1675             )
1676
1677             # Crear el Checkbutton para "Generar archivo PDF"
1678             checkbutton_3 = tk.Checkbutton(
1679                 labelFrame_complementos,
1680                 text="Generar archivo PDF",
1681                 variable=var2,
1682                 onvalue=1,
1683                 offvalue=0,
1684                 indicatoron=True,
1685                 selectcolor="",
```

```
1686         command=lambda: cambiar_color(checkbutton_3, var2)
1687     )
1688
1689
1690     # Crear el Checkbutton para "Exportar datos a Excel"
1691     checkbutton_4 = tk.Checkbutton(
1692         labelFrame_complementos,
1693         text="Exportar datos a Excel",
1694         variable=var3,
1695         onvalue=1,
1696         offvalue=0,
1697         indicatoron=True,
1698         selectcolor="",
1699         command=lambda: cambiar_color(checkbutton_4, var3)
1700     )
1701     # Crear el Checkbutton para seleccionar formato predefinido
1702     checkbutton_5 = tk.Checkbutton(
1703         label_formato_unidades,
1704         text="Seleccionar formato predefinido",
1705         variable=var4,
1706         onvalue=1,
1707         offvalue=0,
1708         indicatoron=True,
1709         selectcolor="",
1710         command=lambda: [aplicar_formato_predefinido(),
cambiar_color(checkbutton_5, var4)]
1711     )
1712     # Crear el Checkbutton para seleccionar formato predefinido
1713     checkbutton_6 = tk.Checkbutton(
1714         labelFrame_Esfuerzos,
1715         text="Seleccionar configuración por defecto",
1716         variable=var5,
1717         onvalue=1,
1718         offvalue=0,
1719         indicatoron=True,
1720         selectcolor="",
1721         command=lambda: [cambiar_a_valores_por_defecto(),
cambiar_color(checkbutton_6, var5)]
1722     )
1723
1724     def actualizar_fecha():
1725         # Obtener la fecha actual en la zona horaria de Colombia
1726         zona_horaria_colombia = timezone('America/Bogota')
1727         fecha_actual = datetime.now(zona_horaria_colombia).strftime('%d-%m-
%Y')
1728
1729         # Verificar si el widget sigue existiendo antes de actualizarlo
1730         if entry_fecha_entrada.winfo_exists(): # Verifica si el widget sigue
existiendo
1731             # Limpiar el Entry y poner la nueva fecha
1732             entry_fecha_entrada.delete(0, tk.END)
1733             entry_fecha_entrada.insert(0, fecha_actual)
1734
1735         # Llamar a esta función de nuevo después de 1000 ms (1 segundo) para
actualizar la fecha
1736         ventana.after(1000, actualizar_fecha)
1737
1738         # Llamar a la función para actualizar la fecha
1739         actualizar_fecha()
1740
```

```
1741     # Función para obtener la ruta de la carpeta "Archivos_Pilot" en el
escritorio
1742     def obtener_ruta_archivos_pilot():
1743     # Obtenemos el directorio de "Escritorio" según el sistema operativo
1744         sistema = platform.system()
1745
1746         if sistema == "Windows":
1747     # En Windows, obtenemos el "Desktop" del usuario
1748             ruta_base = os.path.join(os.environ["USERPROFILE"], "Desktop")
1749         elif sistema == "Darwin": # macOS
1750     # En macOS, obtenemos el "Desktop" del usuario
1751             ruta_base = os.path.join(os.path.expanduser("~"), "Desktop")
1752         elif sistema == "Linux":
1753     # En Linux, igualmente el "Desktop"
1754             ruta_base = os.path.join(os.path.expanduser("~"), "Desktop")
1755         else:
1756     # Si no es Windows, macOS ni Linux, retornamos una ruta genérica
1757             ruta_base = os.path.expanduser("~")
1758
1759     # Ruta final donde guardaremos los archivos
1760             ruta_archivos_pilot = os.path.join(ruta_base, "Archivos_Pilot")
1761
1762     # Si la carpeta "Archivos_Pilot" no existe, la creamos
1763             if not os.path.exists(ruta_archivos_pilot):
1764                 os.makedirs(ruta_archivos_pilot)
1765
1766             return ruta_archivos_pilot
1767
1768     # Función para mostrar la ruta en el Entry
1769     def mostrar_ruta_archivos_pilot():
1770     # Obtener la ruta de "Archivos_Pilot" dependiendo del sistema
1771         ruta_archivos = obtener_ruta_archivos_pilot()
1772
1773     # Mostrar la ruta en el Entry
1774         entry_ruta_archivo.delete(0, tk.END) # Limpiar el Entry
1775         entry_ruta_archivo.insert(0, ruta_archivos) # Insertar la ruta
correcta
1776
1777     # Llamamos a la función para mostrar la ruta al iniciar la aplicación
1778     mostrar_ruta_archivos_pilot()
1779
1780
1781     # Ruta de la carpeta base_datos en tu entorno de programa
1782     carpeta_base_datos = "base_datos"
1783     archivo_datos_materiales = os.path.join(carpeta_base_datos,
"datos_materiales.txt") # Archivo donde se guardarán los datos
1784
1785     # Crear la carpeta base_datos si no existe
1786     if not os.path.exists(carpeta_base_datos):
1787         os.makedirs(carpeta_base_datos)
1788
1789     ventana_materiales_abierta = False
1790     # Variables globales para los Entry y ComboBox
1791     entry_resistencia_concreto = None
1792     entry_modulo_concreto = None
1793     combo_resistencia_concreto = None
1794     label_mensaje_materiales = None
1795
1796     # Función para guardar los datos de los materiales
1797     def guardar_datos_materiales():
```

```
1798     """Guarda los datos de los widgets en un archivo de texto (ventana
materiales)."""
1799     # Obtener los datos de los Entry y ComboBox
1800     resistencia_concreto = entry_resistencia_concreto.get().strip()
1801     modulo_concreto = entry_modulo_concreto.get().strip()
1802     #Obtener el texto del Label (mensaje)
1803     mensaje_materiales = label_mensaje_materiales.cget("text")
1804
1805
1806     # Crear un diccionario con los datos que se van a guardar
1807     datos_materiales = {
1808         "resistencia_concreto": resistencia_concreto,
1809         "modulo_concreto": modulo_concreto,
1810         "mensaje_materiales": mensaje_materiales
1811     }
1812
1813     # Guardar los datos en el archivo de texto
1814     with open(archivo_datos_materiales, "w") as archivo:
1815         for clave, valor in datos_materiales.items():
1816             archivo.write(f"{clave}:{valor}\n")
1817
1818
1819
1820 # Función para cargar los datos desde el archivo de materiales
1821 def cargar_datos_materiales():
1822     """Carga los datos desde el archivo de texto de materiales si existe."""
1823     datos_materiales = {}
1824     try:
1825         with open(archivo_datos_materiales, "r") as archivo:
1826             for linea in archivo:
1827                 clave, valor = linea.strip().split(":", 1)
1828                 datos_materiales[clave] = valor
1829     except FileNotFoundError:
1830         pass # Si el archivo no existe, no hacemos nada
1831
1832     # Cargar los datos en los Entry de la ventana de materiales y label
1833     if datos_materiales:
1834         entry_resistencia_concreto.insert(0,
datos_materiales.get("resistencia_concreto", ""))
1835         entry_modulo_concreto.insert(0, datos_materiales.get("modulo_concreto",
""))
1836         # Cargar el mensaje en el Label
1837
1838         label_mensaje_materiales.config(text=datos_materiales.get("mensaje_materiales",
""))
1839
1840 # Función para manejar la lógica de la ventana de materiales
1841 def on_aceptar_materiales():
1842     """Función que se ejecuta cuando se hace clic en el botón 'Aceptar'"""
1843     # Guardar los datos de los Entry y ComboBox en el archivo
1844     guardar_datos_materiales()
1845
1846     # Aquí puedes agregar más lógica si deseas, como cerrar la ventana o realizar
otro proceso
1847
1848     # Cerrar la ventana de materiales
1849     global ventana_materiales_abierta, ventana_materiales
1850     ventana_materiales_abierta = False # Marcar que la ventana ya no está abierta
1851     ventana_materiales.destroy() # Cerrar la ventana de materiales
```

```

1852 # Función para abrir la ventana materiales
1853
1854 def abrir_ventana_materiales():
1855     global ventana_materiales_abierta, ventana_materiales
1856     global entry_resistencia_concreto, entry_modulo_concreto,
1857     combo_resistencia_concreto, label_mensaje_materiales, label_elasticidad, Ec
1858     global label_resistencia_psi
1859
1860     # Verificar si la ventana ya está abierta
1861     if ventana_materiales_abierta:
1862         return # Si ya está abierta, no hacer nada
1863
1864     # Crear una nueva ventana (Toplevel)
1865     ventana_materiales = tk.Toplevel()
1866     ventana_materiales.title("Materiales")
1867     ventana_materiales.geometry("600x600") # Ajusta el tamaño según sea necesario
1868
1869     # Bloquear la opción de redimensionar la ventana
1870     ventana_materiales.resizable(False, False)
1871
1872     # Mantener la ventana siempre encima
1873     ventana_materiales.attributes('-topmost', True)
1874
1875     # Marcar que la ventana está abierta
1876     ventana_materiales_abierta = True
1877
1878     # Lista de datos para el ComboBox
1879     opciones_resistencia = ["17.5 Mpa", "21 Mpa", "24.5 Mpa", "28 Mpa", "35 Mpa",
1880     "42 Mpa"]
1881
1882     # Crear un LabelFrame para agrupar el contenido de Materiales
1883     labelframe_materiales= tk.LabelFrame(ventana_materiales,text="")
1884     labelframe_materiales.pack(padx=20, pady=20, fill="both", expand=True)
1885     labelframe_titulo = tk.LabelFrame(labelframe_materiales,text="Materiales")
1886     labelframe_titulo.place(x=30,y=30,width=500,height=200)
1887     label_resistencia_concreto=tk.Label(labelframe_titulo,text="Resistencia a la
1888     compresión del concreto [f'c]:")
1889     label_resistencia_concreto.place(x=10,y=20)
1890     label_resistencia_tipo = tk.Label(labelframe_titulo,text="Resistencia concreto
1891     tipo [C]:")
1892     label_resistencia_tipo.place(x=10,y=60)
1893     label_resistencia_acero= tk.Label(labelframe_titulo,text="Resistencia a la
1894     tensión del acero [fy]:")
1895     label_resistencia_acero.place(x=10,y=100)
1896     label_peso_concreto = tk.Label(labelframe_titulo,text="Peso específico del
1897     concreto [γ]:")
1898     label_peso_concreto.place(x=10,y=140)
1899     combo_resistencia_concreto= ttk.Combobox(labelframe_titulo,
1900     values=opciones_resistencia)
1901     combo_resistencia_concreto.place(x=300,y=20)
1902     entry_resistencia_concreto= tk.Entry(labelframe_titulo)
1903     entry_resistencia_concreto.place(x=300,y=60,width=143)
1904     entry_resistencia_acero= tk.Entry(labelframe_titulo)
1905     entry_resistencia_acero.place(x=300,y=100,width=143)
1906     entry_resistencia_acero.insert(0, "420 MPa")
1907     entry_peso_concreto= tk.Entry(labelframe_titulo)
1908     entry_peso_concreto.place(x=300,y=140,width=143)
1909     entry_peso_concreto.insert(0, "24.00 kN/m³")
1910     labelframe_propiedades = tk.LabelFrame(labelframe_materiales,text="Propiedades
1911     mecánicas")

```

```

1904     labelframe_propiedades.place(x=30,y=250,width=500,height=200)
1905     label_modulo_concreto = tk.Label(labelframe_propiedades,text="Módulo
elasticidad del concreto [Ec]:")
1906     label_modulo_concreto.place(x=10,y=20)
1907     entry_modulo_concreto= tk.Entry(labelframe_propiedades)
1908     entry_modulo_concreto.place(x=300,y=20,width=143)
1909     label_modulo_acero = tk.Label(labelframe_propiedades,text="Módulo elasticidad
del acero [Es]:")
1910     label_modulo_acero.place(x=10,y=60)
1911     entry_modulo_acero= tk.Entry(labelframe_propiedades)
1912     entry_modulo_acero.place(x=300,y=60,width=143)
1913     entry_modulo_acero.insert(0, "200000 MPa")
1914     label_deformacion_concreto = tk.Label(labelframe_propiedades,text="Deformación
unitaria concreto [εc]: ")
1915     label_deformacion_concreto.place(x=10,y=100)
1916     entry_deformacion_concreto = tk.Entry(labelframe_propiedades)
1917     entry_deformacion_concreto.place(x=300,y=100,width=143)
1918     entry_deformacion_concreto.insert(0, "0.003")
1919     label_deformacion_acero = tk.Label(labelframe_propiedades,text="Deformación
unitaria acero [εs]: ")
1920     label_deformacion_acero.place(x=10,y=140)
1921     entry_deformacion_acero = tk.Entry(labelframe_propiedades)
1922     entry_deformacion_acero.place(x=300,y=140,width=143)
1923     entry_deformacion_acero.insert(0, "0.005")
1924     labelframe_mensaje_materiales= tk.LabelFrame(labelframe_materiales,text="")
1925     labelframe_mensaje_materiales.place(x=30,y=460,width=500,height=50)
1926     label_mensaje_materiales =
tk.Label(labelframe_mensaje_materiales,justify="left",anchor="w",wraplength=500)
1927     label_mensaje_materiales.place(x=10,y=5)
1928     boton_aceptar_materiales =
ttk.Button(labelframe_materiales,text="Aceptar",command=on_aceptar_materiales)
1929     boton_aceptar_materiales.place(x=220,y=520,width=100)
1930
1931     # Cargar datos al abrir la ventana
1932     cargar_datos_materiales()
1933
1934     # Función para actualizar el Entry según la selección del ComboBox
1935     def actualizar_entry(event):
1936         seleccion1 = combo_resistencia_concreto.get() # Obtener la opción
seleccionada
1937         if seleccion1 == "17.5 Mpa":
1938             entry_resistencia_concreto.delete(0, tk.END) # Limpiar el Entry
1939             entry_resistencia_concreto.insert(0, "Concreto 2500 Psi")
1940         elif seleccion1 == "21 Mpa":
1941             entry_resistencia_concreto.delete(0, tk.END)
1942             entry_resistencia_concreto.insert(0, "Concreto 3000 Psi")
1943         elif seleccion1 == "24.5 Mpa":
1944             entry_resistencia_concreto.delete(0, tk.END)
1945             entry_resistencia_concreto.insert(0, "Concreto 3500 Psi")
1946         elif seleccion1 == "28 Mpa":
1947             entry_resistencia_concreto.delete(0, tk.END)
1948             entry_resistencia_concreto.insert(0, "Concreto 4000 Psi")
1949         elif seleccion1 == "35 Mpa":
1950             entry_resistencia_concreto.delete(0, tk.END)
1951             entry_resistencia_concreto.insert(0, "Concreto 5000 Psi")
1952         elif seleccion1 == "42 Mpa":
1953             entry_resistencia_concreto.delete(0, tk.END)
1954             entry_resistencia_concreto.insert(0, "Concreto 6000 Psi")
1955
1956         #cálculo del módulo de elasticidad del concreto (Ec)

```

```
1957     f_c = float(seleccion1.split(" ")[0])
1958     Ec = 3900 * math.sqrt(f_c) # Calcular el módulo de elasticidad
1959     # Actualizar el entry_modulo_concreto con el resultado del cálculo
1960     entry_modulo_concreto.delete(0, tk.END)
1961     entry_modulo_concreto.insert(0, f"{Ec:.2f} MPa")
1962
1963     # Actualizar el Label con el mensaje personalizado
1964     mensaje1 = f"Has seleccionado el material {seleccion1}. Ahora, el siguiente
paso es definir las propiedades geométricas de la sección transversal."
1965     label_mensaje_materiales.config(text=" " + mensaje1)
1966
1967     # Actualizar el label_elasticidad con el valor calculado
1968     label_elasticidad.config(text=f"{Ec:.2f} MPa")
1969
1970     # Obtener el valor de entry_resistencia_concreto y actualizar el
label_resistencia_psi
1971     resistencia_psi = entry_resistencia_concreto.get()
1972     label_resistencia_psi.config(text=f"{resistencia_psi}")
1973
1974     # Asociar la función de actualización del Entry con el evento de selección del
ComboBox
1975     combo_resistencia_concreto.bind("<<ComboboxSelected>>", actualizar_entry)
1976
1977
1978     # Función para manejar el cierre de la ventana
1979     def cerrar_ventana():
1980         global ventana_materiales_abierta
1981         ventana_materiales_abierta = False
1982         ventana_materiales.destroy()
1983
1984     # Asociar el cierre de la ventana con la función `cerrar_ventana`
1985     ventana_materiales.protocol("WM_DELETE_WINDOW", cerrar_ventana)
1986
1987
1988
1989 # Variables globales para los Entry y Labels
1990 entry_luz_seccion = None
1991 entry_base_seccion = None
1992 entry_altura_seccion = None
1993 entry_recubrimiento_seccion = None
1994 entry_d_seccion = None
1995 entry_nombre_seccion = None
1996 label_resumen_1 = None
1997 label_resumen_2 = None
1998 label_resumen_3 = None
1999 label_resumen_4 = None
2000 label_resumen_5 = None
2001 label_resumen_6 = None
2002
2003 # Ruta de la carpeta base_datos en tu entorno de programa
2004 carpeta_base_datos = "base_datos"
2005 archivo_datos_seccion = os.path.join(carpeta_base_datos, "datos_seccion.txt")
2006
2007 # Crear la carpeta base_datos si no existe
2008 if not os.path.exists(carpeta_base_datos):
2009     os.makedirs(carpeta_base_datos)
2010
2011 # Función para guardar los datos de la sección transversal
2012 def guardar_datos_seccion():
```

```
2013     """Guarda los datos de los Entry y Labels en un archivo de texto (ventana
secciones)."""
2014     # Obtener los datos de los Entry y Labels
2015     luz_seccion = entry_luz_seccion.get().strip()
2016     base_seccion = entry_base_seccion.get().strip()
2017     altura_seccion = entry_altura_seccion.get().strip()
2018     recubrimiento_seccion = entry_recubrimiento_seccion.get().strip()
2019     d_seccion = entry_d_seccion.get().strip()
2020     nombre_seccion = entry_nombre_seccion.get().strip()
2021
2022 # Obtener los textos de los Labels (resumen)
2023     resumen_1 = label_resumen_1.cget("text")
2024     resumen_2 = label_resumen_2.cget("text")
2025     resumen_3 = label_resumen_3.cget("text")
2026     resumen_4 = label_resumen_4.cget("text")
2027     resumen_5 = label_resumen_5.cget("text")
2028     resumen_6 = label_resumen_6.cget("text")
2029
2030 # Crear un diccionario con los datos que se van a guardar
2031     datos_seccion = {
2032         "luz_seccion": luz_seccion,
2033         "base_seccion": base_seccion,
2034         "altura_seccion": altura_seccion,
2035         "recubrimiento_seccion": recubrimiento_seccion,
2036         "d_seccion": d_seccion,
2037         "nombre_seccion": nombre_seccion,
2038         "resumen_1": resumen_1,
2039         "resumen_2": resumen_2,
2040         "resumen_3": resumen_3,
2041         "resumen_4": resumen_4,
2042         "resumen_5": resumen_5,
2043         "resumen_6": resumen_6,
2044     }
2045
2046 # Guardar los datos en el archivo de texto
2047     with open(archivo_datos_seccion, "w") as archivo:
2048         for clave, valor in datos_seccion.items():
2049             archivo.write(f"{clave}:{valor}\n")
2050
2051     # Mostrar un mensaje de confirmación
2052     print("Datos guardados exitosamente en el archivo.")
2053
2054 # Función para cargar los datos desde el archivo de la sección
2055 def cargar_datos_seccion():
2056     """Carga los datos desde el archivo de texto de la sección si existe."""
2057     datos_seccion = {}
2058
2059     try:
2060         with open(archivo_datos_seccion, "r") as archivo:
2061             for linea in archivo:
2062                 clave, valor = linea.strip().split(":", 1)
2063                 datos_seccion[clave] = valor
2064     except FileNotFoundError:
2065         pass # Si el archivo no existe, no hacemos nada
2066
2067     # Actualizar los Entry y Labels con los datos cargados
2068     if datos_seccion:
2069         entry_luz_seccion.delete(0, tk.END)
2070         entry_luz_seccion.insert(0, datos_seccion.get("luz_seccion", ""))
2071
```

```
2072     entry_base_seccion.delete(0, tk.END)
2073     entry_base_seccion.insert(0, datos_seccion.get("base_seccion", ""))
2074
2075     entry_altura_seccion.delete(0, tk.END)
2076     entry_altura_seccion.insert(0, datos_seccion.get("altura_seccion", ""))
2077
2078     entry_recubrimiento_seccion.delete(0, tk.END)
2079     entry_recubrimiento_seccion.insert(0,
datos_seccion.get("recubrimiento_seccion", ""))
2080
2081     entry_d_seccion.delete(0, tk.END)
2082     entry_d_seccion.insert(0, datos_seccion.get("d_seccion", ""))
2083
2084     entry_nombre_seccion.delete(0, tk.END)
2085     entry_nombre_seccion.insert(0, datos_seccion.get("nombre_seccion", ""))
2086
2087     # Cargar los valores en los Labels de resumen
2088     label_resumen_1.config(text=datos_seccion.get("resumen_1", ""))
2089     label_resumen_2.config(text=datos_seccion.get("resumen_2", ""))
2090     label_resumen_3.config(text=datos_seccion.get("resumen_3", ""))
2091     label_resumen_4.config(text=datos_seccion.get("resumen_4", ""))
2092     label_resumen_5.config(text=datos_seccion.get("resumen_5", ""))
2093     label_resumen_6.config(text=datos_seccion.get("resumen_6", ""))
2094
2095     # Variable global para controlar si la ventana ya está abierta
2096     ventana_secundaria_abierta = False
2097
2098
2099
2100     canvas_falso = None # Canvas donde dibujaremos el rectángulo en
frame_ventana_falsa
2101     frame_ventana_seccion = None # El frame principal donde se encuentra
frame_ventana_falsa
2102     frame_ventana_falsa = None # El frame donde se dibujará el rectángulo
2103     entry_base_seccion = None # Entry para base del rectángulo
2104     entry_altura_seccion = None # Entry para altura del rectángulo
2105     combo_resistencia = None
2106
2107
2108     # Constantes para grosor
2109     GROSORES = {
2110         'outer_thickness': 1.5, # Grosor del borde externo
2111     }
2112
2113     # Función para dibujar la sección rectangular con base y altura
2114     def draw_section(ax, base, altura):
2115         ax.clear()
2116
2117         # Configuración de estilo
2118         ax.figure.patch.set_facecolor('white')
2119         ax.set_facecolor('white')
2120         for spine in ax.spines.values():
2121             spine.set_edgecolor('black')
2122         ax.xaxis.label.set_color('black')
2123         ax.yaxis.label.set_color('black')
2124         ax.title.set_color('black')
2125         ax.tick_params(axis='both', colors='black')
2126
2127         x = 0
2128         y = 0
```

```

2129     width = base
2130     height = altura
2131
2132     # Dibujo del rectángulo exterior (sección completa)
2133     outer_rect = patches.Rectangle((x, y), width, height,
linewidth=GROSORES['outer_thickness'], edgecolor='black', facecolor='grey')
2134     ax.add_patch(outer_rect)
2135
2136     # Obtener el valor seleccionado del combo_resistencia
2137     resistencia_value = combo_resistencia.get() # Obtener el valor seleccionado
en el combo
2138
2139     # Asegurarse de que entry_luz_seccion esté definido y obtener su valor
2140     luz_value = entry_luz_seccion.get()
2141
2142     # Agregar la leyenda justo debajo de la etiqueta "Base"
2143     legend_line = Line2D([0], [0], color='grey', lw=4, label=f'Concreto
{resistencia_value}')
2144
2145     # Agregar la leyenda para la luz debajo de la anterior
2146     light_line = Line2D([0], [0], color='blue', lw=4, label=f'Luz {luz_value}m')
2147
2148     # Agregar las leyendas y configurar su posición
2149     ax.legend(handles=[legend_line, light_line],
                bbox_to_anchor=(0.5, -0.3), loc='upper center',
2150                fontsize=8, frameon=True, edgecolor='black',
2151                ncol=2) # `ncol=2` coloca las leyendas en dos columnas
2152
2153
2154     # Coordenadas para centrar el título dentro del rectángulo
2155     center_x = base / 2
2156     center_y = altura / 2
2157
2158     # Agregar el texto centrado dentro del rectángulo
2159     ax.text(center_x, center_y, f"VG {int(base)} X {int(altura)}",
                color='black', fontsize=8, ha='center', va='center')
2160
2161
2162     # Configuración de los límites y etiquetas del gráfico
2163     ax.set_xlim(-10, width + 10)
2164     ax.set_ylim(-10, height + 10)
2165     ax.set_aspect('equal')
2166
2167     # Reducir tamaño de los ticks
2168     ax.tick_params(axis='both', which='major', labelsize=6)
2169
2170     # Ajuste automático de los márgenes
2171     plt.tight_layout(pad=0.5)
2172
2173 # Función para actualizar el gráfico con los valores ingresados
2174 def update_plot():
2175     try:
2176         # Obtener los valores de los entries y multiplicarlos por 100
2177         base = float(entry_base_seccion.get().strip()) * 100 # Multiplicamos por
100
2178         altura = float(entry_altura_seccion.get().strip()) * 100 # Multiplicamos
por 100
2179
2180         # Crear la figura y el eje con un tamaño que se ajustará al contenedor
2181         fig, ax = plt.subplots(figsize=(3, 3)) # Tamaño base para la figura
2182         draw_section(ax, base, altura)
2183

```

```

2184 # Actualizar las etiquetas del eje x e y con el nuevo formato
2185 ax.set_xlabel(f"Base ({int(base)}cm)", fontsize=8)
2186 ax.set_ylabel(f"Altura ({int(altura)}cm)", fontsize=8)
2187
2188 # Eliminar los elementos anteriores del canvas
2189 for widget in frame_ventana_falsa.winfo_children():
2190     widget.destroy()
2191
2192 # Crear el canvas
2193 canvas = FigureCanvasTkAgg(fig, master=frame_ventana_falsa)
2194 canvas.draw()
2195
2196 # Obtener el widget del canvas
2197 canvas_widget = canvas.get_tk_widget()
2198
2199 # Crear la etiqueta de mensaje (label) que se animará, solo si no existe
2200 if not hasattr(update_plot, "label_mensaje_falsa"): # Comprobamos si la
etiqueta ya existe
2201     update_plot.label_mensaje_falsa = tk.Label(label_titulo_falsa,
text="¡Muy bien! Ahora, continúa con las opciones de asignar.",
bd=0,relief="flat")
2202     update_plot.label_mensaje_falsa.place(x=1, y=1, width=380, height=20)
2203
2204     # Iniciar la animación de la etiqueta
2205     animate_label_movement(update_plot.label_mensaje_falsa)
2206
2207 # Ajustar el canvas para que ocupe todo el espacio disponible en el
contenedor
2208 canvas_widget.place(relwidth=1, relheight=1) # Ocupa el 100% del espacio
del frame
2209
2210 except ValueError as e:
2211     print(f"Error: {e}")
2212
2213
2214
2215 # Función para animar el movimiento de la etiqueta de mensaje
2216 def animate_label_movement(label):
2217     def move_label():
2218         # Obtener la posición actual del texto
2219         current_x = label.winfo_x()
2220         new_x = current_x + 2 # Desplazar 2 píxeles hacia la derecha
2221
2222         # Si el texto se sale de la pantalla, reiniciarlo desde el borde izquierdo
2223         if new_x > label_titulo_falsa.winfo_width():
2224             new_x = -label.winfo_width()
2225
2226         label.place(x=new_x, y=5) # Actualizar la posición de la etiqueta
2227         label.after(50, move_label) # Llamar a la función cada 50 ms para animar
2228
2229     move_label() # Llamar la función para iniciar la animación
2230
2231
2232 # Función para abrir la ventana seccion transversal
2233
2234 def abrir_ventana_seccion_transversal():
2235     global ventana_secundaria_abierta
2236     global
frame_ventana_seccion,frame_ventana_falsa,label_titulo_falsa,label_mensaje_falsa

```

```
2237 global entry_luz_seccion, entry_base_seccion, entry_altura_seccion,
entry_recubrimiento_seccion, entry_d_seccion, entry_nombre_seccion
2238 global label_resumen_1, label_resumen_2, label_resumen_3, label_resumen_4,
label_resumen_5, label_resumen_6, combo_resistencia
2239 if ventana_secundaria_abierta:
2240     return
2241
2242 # Crear la ventana secundaria
2243 ventana_secundaria = tk.Toplevel()
2244 ventana_secundaria.title("Propiedades geométricas de la sección")
2245
2246 # Evitar que la ventana sea maximizada o redimensionada
2247 ventana_secundaria.resizable(False, False)
2248
2249 # Mantener la ventana siempre encima
2250 ventana_secundaria.attributes('-topmost', True)
2251
2252 # Definir el tamaño de la ventana
2253 ventana_secundaria.geometry("500x530")
2254
2255 # Crear contenido dentro de la ventana secundaria
2256 label_frame_seccion = tk.LabelFrame(ventana_secundaria, text="")
2257 label_frame_seccion.place(x=20,y=20,width=460,height=460)
2258
2259 label_frame_caracteristicas =
tk.LabelFrame(label_frame_seccion,text="Características adicionales")
2260 label_frame_caracteristicas.place(x=30,y=30,width=400,height=80)
2261
2262 label_tipo_viga= tk.Label(label_frame_caracteristicas,text="Elemento tipo:")
2263 label_tipo_viga.place(x=30,y=15)
2264 entry_tipo_viga= tk.Entry(label_frame_caracteristicas)
2265 entry_tipo_viga.place(x=120,y=15 ,width=70)
2266 entry_tipo_viga.insert(0, "Viga")
2267 label_seccion_viga= tk.Label(label_frame_caracteristicas,text="Sección:")
2268 label_seccion_viga.place(x=200,y=15)
2269 entry_seccion_viga = tk.Entry(label_frame_caracteristicas)
2270 entry_seccion_viga.place(x=260,y=15,width=70)
2271 entry_seccion_viga.insert(0,"Rectangular")
2272 frame_seccion_viga= tk.Frame(label_frame_seccion,bd=2,relief="ridge")
2273 frame_seccion_viga.place(x=30,y=120,width=400,height=300)
2274
2275 # Crear un objeto Style para personalizar el estilo del Notebook
2276 style = ttk.Style()
2277
2278 # Cambiar el fondo de las pestañas y del notebook
2279 style.configure("TNotebook", background="lightgrey", relief="flat")
2280 style.configure("TNotebook.Tab", background="lightgrey", foreground="black")
2281
2282 # Cambiar el color del área de contenido dentro de la pestaña
2283 style.map("TNotebook.Tab", background=[("selected", "lightgrey")]) # Color de
la pestaña seleccionada
2284
2285 # Crear un Notebook para las pestañas
2286 notebook = ttk.Notebook(frame_seccion_viga,style="TNotebook")
2287 notebook.pack(fill="both", expand=True)
2288
2289 # Crear la pestaña "Sección transversal"
2290 pestaña_seccion = tk.Frame(notebook)
2291 notebook.add(pestaña_seccion, text="Sección transversal")
2292
```

```

2293 # Crear la pestaña "Resumén"
2294 pestaña_resumen = tk.Frame(notebook)
2295 notebook.add(pestaña_resumen, text="Resumen")
2296
2297 # Crear un Frame dentro de la pestaña "Sección transversal"
2298 frame_pest_seccion = tk.Frame(pestaña_seccion, bd=2, relief="flat",bg="white")
2299 frame_pest_seccion.pack(fill="both", expand=True)
2300
2301 variable_seccion = tk.StringVar()
2302
2303 labelframe_dimensiones= tk.LabelFrame(frame_pest_seccion,text="Dimensiones
[m]",bg="white")
2304 labelframe_dimensiones.place(x=30,y=10,width=330,height=190)
2305 label_luz_seccion = tk.Label(labelframe_dimensiones,text="L:",bg="white")
2306 label_luz_seccion.place(x=10,y=10)
2307 label_base_seccion = tk.Label(labelframe_dimensiones,text="b:",bg="white")
2308 label_base_seccion.place(x=10,y=40)
2309 label_altura_seccion = tk.Label(labelframe_dimensiones,text="h:",bg="white")
2310 label_altura_seccion.place(x=10,y=70)
2311 label_recubrimiento_seccion =
tk.Label(labelframe_dimensiones,text="r:",bg="white")
2312 label_recubrimiento_seccion.place(x=10,y=100)
2313 label_d_seccion = tk.Label(labelframe_dimensiones,text="d:",bg="white")
2314 label_d_seccion.place(x=10,y=130)
2315 label_nombre_seccion = tk.Label(frame_pest_seccion, text="Viga:",bg="white")
2316 label_nombre_seccion.place(x=30,y=220)
2317 entry_nombre_seccion = tk.Entry(frame_pest_seccion,bd=2,
relief="ridge",textvariable=variable_seccion)
2318 entry_nombre_seccion.place(x=80,y=220,width=80)
2319 label_fc_seccion = tk.Label(frame_pest_seccion,text="f'c:",bg="white")
2320 label_fc_seccion.place(x=200,y=220)
2321
2322 # Tarea interna de la función (esto puede ser cualquier acción que modifique
el contenido del Entry)
2323 variable_seccion.set("Algunos datos") # Ejemplo de cómo poner datos en el
Entry
2324
2325 # Ahora el Label fuera de la función puede acceder a la variable_seccion
2326 label_seccion_cargas.config(textvariable=variable_seccion)
2327
2328 # Crear un ComboBox con los valores de resistencia a la compresión
2329 valores_resistencia = [
2330 "17.5 Mpa", "21 Mpa", "24.5 Mpa", "28 Mpa", "35 Mpa", "42 Mpa"
2331 ]
2332 combo_resistencia = ttk.Combobox(frame_pest_seccion,
values=valores_resistencia, state="readonly")
2333 combo_resistencia.place(x=240,y=220,width=80)
2334
2335 frame_pest_resumen = tk.Frame(pestaña_resumen,bd=2,relief="flat",bg="white")
2336 frame_pest_resumen.pack(fill="both", expand=True)
2337
2338 labelframe_resumen = tk.LabelFrame(frame_pest_resumen,bg="white")
2339 labelframe_resumen.place(x=10,y=10,width=370,height=250)
2340 label_resumen = tk.Label(labelframe_resumen,
2341 text="A continuación se presenta un resumen de las
configuraciones utilizadas para definir la sección transversal de la viga.",
2342 justify="left",
2343 anchor="w",
2344 wraplength=350,
2345 bg="white")

```

```
2346     label_resumen.place(x=10,y=10)
2347
2348     label_resumen_nombre = tk.Label(labelframe_resumen,text="Nombre
[VG]:",bg="white")
2349     label_resumen_nombre.place(x=10,y=60)
2350     label_resumen_1 = tk.Label(labelframe_resumen,text="",bg="white")
2351     label_resumen_1.place(x=90,y=60)
2352
2353     label_resumen_luz = tk.Label(labelframe_resumen,text="Longitud
[L]:",bg="white")
2354     label_resumen_luz.place(x=10,y=90)
2355     label_resumen_2 = tk.Label(labelframe_resumen,text="",bg="white")
2356     label_resumen_2.place(x=85,y=90)
2357
2358     label_resumen_base = tk.Label(labelframe_resumen,text="Base [b]:",bg="white")
2359     label_resumen_base.place(x=10,y=120)
2360     label_resumen_3 = tk.Label(labelframe_resumen,text="",bg="white")
2361     label_resumen_3.place(x=65,y=120)
2362
2363     label_resumen_altura = tk.Label(labelframe_resumen,text="Altura
[h]:",bg="white")
2364     label_resumen_altura.place(x=10,y=150)
2365     label_resumen_4 = tk.Label(labelframe_resumen,text="",bg="white")
2366     label_resumen_4.place(x=70,y=150)
2367
2368
2369     label_resumen_recubrimiento = tk.Label(labelframe_resumen,text="Recubrimiento
[r]:",bg="white")
2370     label_resumen_recubrimiento.place(x=10,y=180)
2371     label_resumen_5 = tk.Label(labelframe_resumen,text="",bg="white")
2372     label_resumen_5.place(x=115,y=180)
2373
2374
2375     label_resumen_d = tk.Label(labelframe_resumen,text="Altura efectiva
[d]:",bg="white")
2376     label_resumen_d.place(x=10,y=210)
2377     label_resumen_6 = tk.Label(labelframe_resumen,text="",bg="white")
2378     label_resumen_6.place(x=115,y=210)
2379
2380
2381     def actualizar_label(*args):
2382         # Obtener el valor del Entry
2383         valor_1 = entry_luz_seccion.get()
2384         # Actualizar el Label con el texto fijo + el valor ingresado
2385         label_resumen_2.config(text=f"La viga presenta una luz de {valor_1}
metros.")
2386
2387
2388     def actualizar_label_base(*args):
2389         # Obtener el valor del Entry y actualizar el Label
2390         valor_2 = entry_base_seccion.get()
2391         label_resumen_3.config(text=f"La viga presenta una base de {valor_2}
metros.")
2392
2393
2394
2395         # Función para actualizar d_seccion en tiempo real
2396         def actualizar_d_seccion(*args):
2397             try:
2398                 altura = float(var_altura.get())
```

```
2399     except ValueError:
2400         altura = 0
2401
2402     try:
2403         recubrimiento = float(var_recubrimiento.get())
2404     except ValueError:
2405         recubrimiento = 0
2406
2407     # Realizar la resta
2408     d_seccion = altura - recubrimiento
2409
2410     # Mostrar el resultado en el campo entry_d_seccion
2411     entry_d_seccion.delete(0, tk.END)
2412     entry_d_seccion.insert(0, str(d_seccion))
2413
2414     # Actualizar el texto del label_resumen_5 con el valor de d_seccion
2415     label_resumen_5.config(text=f"El recubrimiento a emplear será de
{recubrimiento * 100:.1f} cm")
2416     # Actualizar el texto del label_resumen_6 con el valor de d_seccion desde
entry_d_seccion
2417     label_resumen_6.config(text=f"La altura efectiva a utilizar es de
{d_seccion:.2f} metros")
2418
2419
2420     def actualizar_nombre_seccion(*args):
2421         # Recuperamos los valores de los Entry
2422         base_nombre = entry_base_seccion.get()
2423         altura_nombre = entry_altura_seccion.get()
2424
2425         try:
2426             # Convertimos los valores a números flotantes y los multiplicamos por 100
2427             base_nombre = round(float(base_nombre) * 100, 2)
2428             altura_nombre = round(float(altura_nombre) * 100, 2)
2429
2430             # Formateamos la cadena con los valores modificados
2431             nombre_seccion = f"VG {base_nombre} X {altura_nombre}"
2432
2433             # Actualizamos el campo entry_nombre_seccion
2434             entry_nombre_seccion.delete(0, tk.END)
2435             entry_nombre_seccion.insert(0, nombre_seccion)
2436
2437             # Actualizamos el texto del label_resumen_1
2438             label_resumen_1.config(text=f"Elemento viga, sección -
{nombre_seccion}")
2439             # Actualizamos el texto del label_resumen_4 con la altura
2440             altura_metros = round(altura_nombre * 0.01, 2)
2441             label_resumen_4.config(text=f"La viga presenta una altura de
{altura_metros} metros")
2442
2443         except ValueError:
2444             # Si los valores no son numéricos, no hacemos nada
2445             pass
2446
2447     # Función para dibujar el rectángulo en el canvas
2448     def dibujar_rectangulo():
2449         try:
2450             base = float(var_base.get())
2451             altura = float(var_altura.get())
2452         except ValueError:
2453             base = 0
```

```

2454         altura = 0
2455
2456         # Definir un factor de escala para que los valores en metros se ajusten al
canvas
2457         factor_escala_base = 120
2458         factor_escala_altura = 142
2459
2460         # Escalar las dimensiones del rectángulo en píxeles
2461         base_pixel = base * factor_escala_base
2462         altura_pixel = altura * factor_escala_altura
2463
2464         # Limpiar el canvas antes de dibujar
2465         canvas_seccion.delete("all")
2466
2467         # Calcular el margen para centrar el rectángulo
2468         canvas_width = 120 # Ancho del Canvas
2469         canvas_height = 142 # Altura del Canvas
2470         x_offset = (canvas_width - base_pixel) / 2 - 5
2471         y_offset = (canvas_height - altura_pixel) / 2
2472
2473         # Dibujar el rectángulo en el canvas
2474         canvas_seccion.create_rectangle(
2475             x_offset, y_offset, x_offset + base_pixel, y_offset + altura_pixel,
2476             outline="black", fill="medium blue", width=1
2477         )
2478         # Calcular el valor escalado de base y altura (multiplicados por 100)
2479         base_multiplicada = base * 100
2480         altura_multiplicada = altura * 100
2481
2482         # Agregar el título centrado arriba del rectángulo
2483         texto_titulo = f"VG {base_multiplicada:.0f} X {altura_multiplicada:.0f}"
2484         canvas_seccion.create_text(
2485             (canvas_width / 2)-6, y_offset - 20,
2486             text=texto_titulo,
2487             font=("Arial", 8, "bold"),
2488             fill="black"
2489         )
2490         # Agregar las cotas a la base (en la parte inferior central del
rectángulo)
2491         canvas_seccion.create_text(
2492             x_offset + base_pixel / 2, y_offset + altura_pixel + 10,
2493             text=f"{base:.2f} m",
2494             font=("Arial", 8, "bold"),
2495             fill="black"
2496         )
2497
2498         # Agregar las cotas a la altura (en la parte izquierda central del
rectángulo)
2499         canvas_seccion.create_text(
2500             x_offset -8, y_offset + altura_pixel / 2,
2501             text=f"{altura:.2f} m",
2502             font=("Arial", 8, "bold"),
2503             fill="black",
2504             angle=90
2505         )
2506
2507
2508         # Crear los StringVar para monitorizar cambios en los Entry
2509         var_base = tk.StringVar()
2510         var_altura = tk.StringVar()

```

```
2511     var_recubrimiento = tk.StringVar()
2512
2513
2514     entry_luz_seccion = tk.Entry(labelframe_dimensiones,bd=2, relief="ridge")
2515     entry_luz_seccion.place(x=50,y=10,width=50)
2516     entry_base_seccion = tk.Entry(labelframe_dimensiones,bd=2, relief="ridge")
2517     entry_base_seccion.place(x=50,y=40,width=50)
2518     entry_base_seccion.config(textvariable=var_base)
2519     entry_altura_seccion = tk.Entry(labelframe_dimensiones,bd=2, relief="ridge")
2520     entry_altura_seccion.place(x=50,y=70,width=50)
2521     entry_altura_seccion.config(textvariable=var_altura)
2522     entry_recubrimiento_seccion = tk.Entry(labelframe_dimensiones,bd=2,
relief="ridge")
2523     entry_recubrimiento_seccion.place(x=50,y=100,width=50)
2524     entry_recubrimiento_seccion.config(textvariable=var_recubrimiento)
2525     entry_d_seccion = tk.Entry(labelframe_dimensiones,bd=2,relief="ridge")
2526     entry_d_seccion.place(x=50,y=130,width=50)
2527     frame_dibujo_seccion = tk.Frame(labelframe_dimensiones,bd=4,relief="ridge")
2528     frame_dibujo_seccion.place(x=170,y=10,width=120,height=142)
2529     # Crear el Canvas dentro del frame_dibujo_seccion
2530     canvas_seccion = tk.Canvas(frame_dibujo_seccion, bg="lightgrey", width=120,
height=142)
2531     canvas_seccion.pack(fill="both", expand=True)
2532
2533     #Crear el boton aceptar para guardar datos
2534     boton_aceptar_seccion=
ttk.Button(ventana_secundaria,text="Aceptar",command=guardar_datos_seccion)
2535     boton_aceptar_seccion.place(x=100,y=490,width=100)
2536
2537     # Cargar los datos guardados al abrir la ventana
2538     cargar_datos_seccion()
2539
2540
2541     #Crar el boton aplicar para mostrar grafico dinamico
2542     boton_aplicar_seccion=
ttk.Button(ventana_secundaria,text="Aplicar",command=update_plot)
2543     boton_aplicar_seccion.place(x=280,y=490,width=100)
2544
2545
2546     # Agregar el trace a las variables para actualizar en tiempo real
2547     var_base.trace("w", lambda *args: dibujar_rectangulo())
2548     var_altura.trace("w", lambda *args: dibujar_rectangulo())
2549
2550
2551     # Agregar el trace a las variables para actualizar en tiempo real
2552     var_altura.trace("w", actualizar_d_seccion)
2553     var_recubrimiento.trace("w", actualizar_d_seccion)
2554
2555     # Asociar el cambio de valor de los Entry a la función de actualización
2556     entry_base_seccion.bind("<KeyRelease>", actualizar_nombre_seccion)
2557     entry_altura_seccion.bind("<KeyRelease>", actualizar_nombre_seccion)
2558
2559     # Configurar un evento que se active cada vez que el valor del Entry cambie
2560     entry_luz_seccion.bind("<KeyRelease>", actualizar_label)
2561
2562     # Configurar un evento que se active cada vez que el valor del Entry cambie
2563     entry_base_seccion.bind("<KeyRelease>", actualizar_label_base)
2564
2565     # Marcar la ventana como abierta
2566     ventana_secundaria_abierta = True
```

```
2567
2568
2569     # Configurar para que cuando se cierre la ventana, se pueda volver a abrir
2570     ventana_secundaria.protocol("WM_DELETE_WINDOW", lambda:
cerrar_ventana(ventana_secundaria))
2571
2572     def cerrar_ventana(ventana_secundaria):
2573         global ventana_secundaria_abierta
2574         ventana_secundaria.destroy()
2575         ventana_secundaria_abierta = False
2576
2577 # Variables globales
2578 ventana_apoyos_abierta = False
2579 ventana_apoyos = None
2580 labelframe_apoyos = None
2581 combo_tipo_apoyo = None
2582 label_restriccion_apoyo = None
2583
2584 # Función para cerrar la ventana de apoyos y actualizar la variable
2585 def cerrar_ventana_apoyos():
2586     global ventana_apoyos_abierta, ventana_apoyos
2587     ventana_apoyos_abierta = False
2588     if ventana_apoyos:
2589         ventana_apoyos.destroy()
2590         ventana_apoyos = None
2591
2592 # Función para actualizar los RadioButtons en función del tipo de apoyo
seleccionado
2593 def actualizar_radio_buttons(event):
2594     global labelframe_apoyos, combo_tipo_apoyo # Acceder a las variables globales
2595
2596     # Limpiar los RadioButtons previos
2597     for widget in labelframe_apoyos.winfo_children():
2598         widget.destroy()
2599
2600     tipo_apoyo = combo_tipo_apoyo.get() # Obtener el valor seleccionado en el
ComboBox
2601
2602     # Actualizar el texto del label con el tipo de apoyo seleccionado
2603     label_restriccion_apoyo.config(text=f"{tipo_apoyo}")
2604
2605     if tipo_apoyo == "Apoyo móvil":
2606         # Mostrar los RadioButtons para el apoyo móvil
2607         radio_desplazamiento_y = tk.Radiobutton(labelframe_apoyos, text="No
permite desplazamiento vertical")
2608         radio_desplazamiento_y.grid(row=0, column=0, sticky="w", padx=10, pady=5)
2609
2610         radio_rotacion = tk.Radiobutton(labelframe_apoyos, text="No permite
rotación")
2611         radio_rotacion.grid(row=1, column=0, sticky="w", padx=10, pady=5)
2612
2613         radio_resistencia_y = tk.Radiobutton(labelframe_apoyos, text="Sí permite
movimiento horizontal")
2614         radio_resistencia_y.grid(row=2, column=0, sticky="w", padx=10, pady=5)
2615
2616     elif tipo_apoyo == "Apoyo empotrado":
2617         # Mostrar los RadioButtons para el apoyo empotrado
2618         radio_no_desplazamiento = tk.Radiobutton(labelframe_apoyos, text="No
permite desplazamiento en ninguna dirección")
2619         radio_no_desplazamiento.grid(row=0, column=0, sticky="w", padx=10, pady=5)
```

```
2620
2621     radio_resistencia_vertical = tk.Radiobutton(labelframe_apoyos,
text="Resistencia a fuerzas verticales")
2622     radio_resistencia_vertical.grid(row=1, column=0, sticky="w", padx=10,
pady=5)
2623
2624     radio_resistencia_horizontal = tk.Radiobutton(labelframe_apoyos,
text="Resistencia a fuerzas horizontales")
2625     radio_resistencia_horizontal.grid(row=2, column=0, sticky="w", padx=10,
pady=5)
2626
2627     radio_resistencia_momentos = tk.Radiobutton(labelframe_apoyos,
text="Resistencia a momentos")
2628     radio_resistencia_momentos.grid(row=3, column=0, sticky="w", padx=10,
pady=5)
2629
2630
2631 # Función para abrir la ventana de apoyos
2632 def abrir_ventana_apoyos():
2633     global ventana_apoyos_abierta, ventana_apoyos, labelframe_apoyos,
combo_tipo_apoyo, label_restriccion_apoyo
2634
2635     # Si la ventana ya está abierta, no hacemos nada
2636     if ventana_apoyos_abierta:
2637         return
2638
2639     # Crear la ventana secundaria
2640     ventana_apoyos = tk.Toplevel()
2641     ventana_apoyos.title("Definir apoyos")
2642
2643     # Evitar que la ventana sea maximizada o redimensionada
2644     ventana_apoyos.resizable(False, False)
2645
2646     # Mantener la ventana siempre encima
2647     ventana_apoyos.attributes('-topmost', True)
2648
2649     # Definir el tamaño de la ventana
2650     ventana_apoyos.geometry("500x255")
2651
2652     # Actualizar el estado de la ventana
2653     ventana_apoyos_abierta = True
2654
2655     # Crear contenido en la venta_apoyos
2656     label_tipo_apoyo =tk.Label(ventana_apoyos,text="Tipo de apoyo:")
2657     label_tipo_apoyo.place(x=80,y=20)
2658     labelframe_apoyos = tk.LabelFrame(ventana_apoyos,text="Restricciones de
apoyo")
2659     labelframe_apoyos.place(x=20,y=50,width=465,height=160)
2660
2661
2662     #Crear combobox para tipo de apoyo
2663     # Lista de opciones para el ComboBox
2664     tipos_de_apoyo = ["Apoyo móvil", "Apoyo empotrado"]
2665     combo_tipo_apoyo = ttk.Combobox(ventana_apoyos, values=tipos_de_apoyo,
state="readonly")
2666     combo_tipo_apoyo.place(x=200,y=20)
2667     combo_tipo_apoyo.set("Seleccionar")
2668
2669     # Asignar el manejador de cambio de selección en el ComboBox
2670     combo_tipo_apoyo.bind("<<ComboboxSelected>>", actualizar_radio_buttons)
```

```
2671
2672     # Agregar botón "Aceptar" que cierra la ventana
2673     boton_aceptar_apoyo = ttk.Button(ventana_apoyos, text="Aceptar",
command=cerrar_ventana_apoyos)
2674     boton_aceptar_apoyo.place(x=200, y=220, width=100)
2675
2676     # Asignar el manejador de cierre de la ventana
2677     ventana_apoyos.protocol("WM_DELETE_WINDOW", cerrar_ventana_apoyos)
2678
2679
2680 label_restriccion_apoyo = None
2681 entry_base_cargas = None
2682 entry_altura_cargas = None
2683 ventana_cargas_abierta = None
2684 combo_seccion_analisis = None
2685 entry_Wpp_cargas = None
2686
2687
2688 def calcular_Wpp(*args):
2689     try:
2690         # Obteniendo los valores de las entradas y convirtiéndolos a flotantes
2691         base_wpp = float(entry_base_cargas.get())
2692         altura_wpp = float(entry_altura_cargas.get())
2693
2694         # Calculando Wpp
2695         Wpp_1 = base_wpp * altura_wpp * 24.00
2696
2697         # Mostrando el resultado en el entry_Wpp_cargas
2698         entry_Wpp_cargas.delete(0, tk.END)
2699         entry_Wpp_cargas.insert(0, f"{Wpp_1:.2f}")
2700         # Actualizar el label con el mismo valor calculado
2701         label_cargas_Wpp.config(text=f"{Wpp_1:.2f} kN/m")
2702     except ValueError:
2703
2704         entry_Wpp_cargas.delete(0, tk.END)
2705         label_cargas_Wpp.config(text="Error en los valores")
2706
2707 def calcular_Wtotal(*args):
2708     try:
2709         # Obtener los valores de los Entry
2710         WD = float(entry_WD_cargas.get() or 0) # Si está vacío, se toma como 0
2711         WL = float(entry_WL_cargas.get() or 0) # Si está vacío, se toma como 0
2712         Wpp = float(entry_Wpp_cargas.get() or 0) # Este ya es calculado
2713
internamente
2714
2715         # Calcular Wtotal
2716         Wtotal = WD + WL + Wpp
2717
2718         # Mostrar el resultado en el entry_Wtotal_cargas
2719         entry_Wtotal_cargas.delete(0, tk.END)
2720         entry_Wtotal_cargas.insert(0, f"{Wtotal:.2f}") # Formato con dos
decimales
2721
2722         # Actualizar el label de cada valor (WD, WL, Wtotal)
2723         label_cargas_WD.config(text=f"{WD:.2f} kN/m")
2724         label_cargas_WL.config(text=f"{WL:.2f} kN/m")
2725         label_cargas_Wtotal.config(text=f"{Wtotal:.2f} kN/m")
2726
2727         # Calcular Wu después de Wtotal
2728         calcular_Wu()
```

```

2728 except ValueError:
2729     # Si hay un error en la conversión, limpiar el entry_Wtotal_cargas
2730     entry_Wtotal_cargas.delete(0, tk.END)
2731     label_cargas_WD.config(text="Error en los valores")
2732     label_cargas_WL.config(text="Error en los valores")
2733     label_cargas_Wtotal.config(text="Error en los valores")
2734
2735 # Dentro de la función calcular_Wu, asegúrate de que Wu_var se actualice
correctamente
2736 def calcular_Wu(*args):
2737     try:
2738         # Obtener los valores de los Entry
2739         WD = float(entry_WD_cargas.get() or 0) # Si está vacío, se toma como 0
2740         WL = float(entry_WL_cargas.get() or 0) # Si está vacío, se toma como 0
2741         Wpp = float(entry_Wpp_cargas.get() or 0) # Este ya es calculado
internamente
2742
2743         # Calcular Wu
2744         Wu_carga = 1.2 * (WD + Wpp) + 1.6 * WL
2745
2746         # Mostrar el resultado en el entry_Wu_cargas
2747         entry_Wu_cargas.delete(0, tk.END)
2748         entry_Wu_cargas.insert(0, f"{Wu_carga:.2f}") # Formato con dos decimales
2749
2750         # Actualizar Wu_var con el nuevo valor calculado
2751         Wu_var.set(f"{Wu_carga:.2f}") # Esto actualiza la variable vinculada
2752
2753 except ValueError:
2754     # Si hay un error en la conversión, limpiar el entry_Wu_cargas
2755     entry_Wu_cargas.delete(0, tk.END)
2756     Wu_var.set("0") # Opcional: Resetear Wu_var si hay un error
2757
2758 def actualizar_entries(event):
2759     # Obtener la opción seleccionada en el ComboBox
2760     seleccion_1 = combo_seccion_analisis.get()
2761
2762     try:
2763         # Separar la cadena seleccionada
2764         partes = seleccion_1.replace("VG", "").replace("X", "").split()
2765
2766         # Convertir los valores a flotantes y dividir entre 100
2767         base_valor = float(partes[0]) / 100
2768         altura_valor = float(partes[1]) / 100
2769
2770         # Actualizar los valores en los Entry
2771         entry_base_cargas.delete(0, tk.END)
2772         entry_base_cargas.insert(0, f"{base_valor:.2f}")
2773
2774         entry_altura_cargas.delete(0, tk.END)
2775         entry_altura_cargas.insert(0, f"{altura_valor:.2f}")
2776
2777         # Recalcular Wpp
2778         calcular_Wpp()
2779     except (ValueError, IndexError):
2780         print("Error: El formato de la opción seleccionada no es correcto.")
2781
2782
2783 def actualizar_combo_seccion(combo_seccion_analisis):
2784     # Obtener el valor de label_seccion_cargas
2785     valor_seccion = label_seccion_cargas.cget("text")

```

```
2786
2787     # Si el valor no está vacío, actualizar el ComboBox con ese valor
2788     if valor_seccion:
2789         combo_seccion_analisis['values'] = ("Seleccionar", valor_seccion)
2790     else:
2791         # Si el valor está vacío, solo mostrar la opción "Seleccionar"
2792         combo_seccion_analisis['values'] = ("Seleccionar",)
2793
2794 def dibujar_viga_apoyada_1(canvas_viga_analisis, valor_carga, valor_Wu):
2795     # Obtener el tamaño del canvas
2796     ancho_canvas = canvas_viga_analisis.winfo_width()
2797     alto_canvas = canvas_viga_analisis.winfo_height()
2798
2799     # Calcular la posición central para la viga
2800     x_inicio = ancho_canvas // 4
2801     x_fin = 3 * ancho_canvas // 4
2802     y_viga = alto_canvas // 2
2803
2804     # Dibujar la viga (línea horizontal) centrada en el canvas
2805     canvas_viga_analisis.create_line(x_inicio, y_viga, x_fin, y_viga, width=3,
fill="black")
2806
2807     # Dibujar los triángulos debajo de la línea, con la punta hacia abajo
2808     # Triángulo izquierdo
2809     canvas_viga_analisis.create_polygon(x_inicio, y_viga, x_inicio - 10, y_viga +
20, x_inicio + 10, y_viga + 20, fill="blue", outline="black")
2810
2811     # Triángulo derecho
2812     canvas_viga_analisis.create_polygon(x_fin, y_viga, x_fin - 10, y_viga + 20,
x_fin + 10, y_viga + 20, fill="blue", outline="black")
2813
2814     # Obtener el valor de carga desde el Entry
2815     valor_carga_texto = valor_carga if valor_carga else "0"
2816
2817     # Dibujar la cota (dimensión de la viga) con flechas en ambos extremos
2818     canvas_viga_analisis.create_line(x_inicio, y_viga + 30, x_fin, y_viga + 30,
width=1, fill="red") # Línea de la cota (más fina)
2819     canvas_viga_analisis.create_text((x_inicio + x_fin) / 2, y_viga + 40,
text=f"Longitud: {valor_carga_texto} m ", font=("Arial", 10, "bold"), fill="red")
# Longitud de la viga
2820
2821
2822     # Dibujar las flechas en ambos extremos de la línea de cota (más pequeñas)
2823     canvas_viga_analisis.create_polygon(x_inicio - 3, y_viga + 30, x_inicio + 3,
y_viga + 25, x_inicio + 3, y_viga + 35, fill="red", outline="red")
2824     canvas_viga_analisis.create_polygon(x_fin + 3, y_viga + 30, x_fin - 3, y_viga
+ 25, x_fin - 3, y_viga + 35, fill="red", outline="red")
2825
2826     # Dibujar la línea verde horizontal justo encima de las líneas de carga
distribuida
2827     canvas_viga_analisis.create_line(x_inicio, y_viga - 20, x_fin, y_viga - 20,
width=2, fill="green") # Línea verde horizontal
2828
2829     # Dibujar la carga distribuida a lo largo de la viga (líneas cortas con
flechas hacia abajo)
2830     num_lineas = 20 # Número de segmentos para la carga distribuida
2831     carga_espaciado = (x_fin - x_inicio) / (num_lineas - 1) # Espaciado entre las
líneas de carga
2832
2833     for i in range(num_lineas):
```

```

2834     x_pos = x_inicio + i * carga_espaciado
2835     # Dibujar línea de carga distribuida
2836     canvas_viga_analisis.create_line(x_pos, y_viga - 10, x_pos, y_viga - 20,
width=2, fill="green") # Línea de carga distribuida
2837
2838     # Dibujar la flecha apuntando hacia abajo (ubicada justo debajo de la
línea verde)
2839     canvas_viga_analisis.create_polygon(x_pos - 3, y_viga - 10, x_pos + 3,
y_viga - 10, x_pos, y_viga, fill="green", outline="green")
2840
2841     # Eliminar el texto anterior (si existe) para actualizar la carga distribuida
2842     canvas_viga_analisis.delete("carga_Wu_texto") # Borrar el texto viejo
2843
2844     # Agregar texto a la carga distribuida
2845     canvas_viga_analisis.create_text((x_inicio + x_fin) / 2, y_viga - 40,
text=f"Carga Wu: {valor_Wu} kN/m", font=("Arial", 10, "bold"), fill="green")
2846
2847
2848 def dibujar_viga_empotrada_1(canvas_viga_analisis, valor_carga, valor_Wu):
2849     # Obtener el tamaño del canvas
2850     ancho_canvas = canvas_viga_analisis.winfo_width()
2851     alto_canvas = canvas_viga_analisis.winfo_height()
2852
2853     # Calcular la posición central para la viga
2854     x_inicio = ancho_canvas // 4
2855     x_fin = 3 * ancho_canvas // 4
2856     y_viga = alto_canvas // 2
2857
2858     # Dibujar la viga (línea horizontal) centrada en el canvas
2859     canvas_viga_analisis.create_line(x_inicio, y_viga, x_fin, y_viga, width=3,
fill="black")
2860
2861     # Dibujar las líneas verticales en los extremos (empotradas)
2862     canvas_viga_analisis.create_line(x_inicio, y_viga - 20, x_inicio, y_viga + 20,
width=3, fill="blue")
2863     canvas_viga_analisis.create_line(x_fin, y_viga - 20, x_fin, y_viga + 20,
width=3, fill="blue")
2864
2865     # Obtener el valor de carga desde el Entry
2866     valor_carga_texto = valor_carga if valor_carga else "0"
2867
2868     # Dibujar la cota (dimensión de la viga) con flechas en ambos extremos
2869     canvas_viga_analisis.create_line(x_inicio, y_viga + 30, x_fin, y_viga + 30,
width=1, fill="red") # Línea de la cota (más fina)
2870     canvas_viga_analisis.create_text((x_inicio + x_fin) / 2, y_viga + 40,
text=f"Longitud: {valor_carga_texto} m ", font=("Arial", 10, "bold"), fill="red")
    # Longitud de la viga
2871
2872     # Dibujar las flechas en ambos extremos de la línea de cota (más pequeñas)
2873     canvas_viga_analisis.create_polygon(x_inicio - 3, y_viga + 30, x_inicio + 3,
y_viga + 25, x_inicio + 3, y_viga + 35, fill="red", outline="red")
2874     canvas_viga_analisis.create_polygon(x_fin + 3, y_viga + 30, x_fin - 3, y_viga
+ 25, x_fin - 3, y_viga + 35, fill="red", outline="red")
2875
2876     # Dibujar la carga distribuida a lo largo de la viga (líneas cortas con
flechas hacia abajo)
2877     num_lineas = 20 # Número de segmentos para la carga distribuida
2878     carga_espaciado = (x_fin - x_inicio) / (num_lineas - 1) # Espaciado entre las
líneas de carga
2879

```

```

2880     for i in range(1, num_lineas - 1): # Excluir la primera y la última línea
2881         x_pos = x_inicio + i * carga_espaciado
2882         # Dibujar línea de carga distribuida
2883         canvas_viga_analisis.create_line(x_pos, y_viga - 10, x_pos, y_viga - 20,
width=2, fill="green") # Línea de carga distribuida
2884
2885         # Dibujar la flecha apuntando hacia abajo (ubicada justo debajo de la
línea verde)
2886         canvas_viga_analisis.create_polygon(x_pos - 3, y_viga - 10, x_pos + 3,
y_viga - 10, x_pos, y_viga, fill="green", outline="green")
2887
2888         # Dibujar la línea verde horizontal justo encima de las líneas de carga
distribuida
2889         canvas_viga_analisis.create_line(x_inicio + carga_espaciado, y_viga - 20,
x_fin - carga_espaciado, y_viga - 20, width=2, fill="green")
2890
2891         # Eliminar el texto anterior (si existe) para actualizar la carga distribuida
2892         canvas_viga_analisis.delete("carga_Wu_texto") # Borrar el texto viejo
2893
2894         # Agregar texto a la carga distribuida
2895         canvas_viga_analisis.create_text((x_inicio + x_fin) / 2, y_viga - 40,
text=f"Carga Wu: {valor_Wu} kN/m", font=("Arial", 10, "bold"), fill="green")
2896
2897
2898 def actualizar_diagrama_viga(event, combo_viga_cargas, canvas_viga_analisis,
entry_luz_cargas, Wu_var, label_tipo_viga_cargas):
2899     if not canvas_viga_analisis.winfo_exists():
2900         print("El Canvas no existe.")
2901         return
2902     # Obtener el valor ingresado en el Entry
2903     valor_carga = entry_luz_cargas.get()
2904
2905     # Obtener el valor calculado de la carga distribuida desde Wu_var (la variable
vinculada al entry)
2906     valor_Wu = Wu_var.get() # Este es el valor calculado de Wu
2907
2908     # Verificar la opción seleccionada en el ComboBox
2909     seleccion_1 = combo_viga_cargas.get()
2910
2911     # Actualizar el texto del label con el tipo de viga seleccionado
2912     label_tipo_viga_cargas.config(text=seleccion_1)
2913
2914     # Limpiar el canvas antes de dibujar un nuevo diagrama
2915     canvas_viga_analisis.delete("all")
2916
2917     if seleccion_1 == "Viga apoyada":
2918         dibujar_viga_apoyada_1(canvas_viga_analisis, valor_carga, valor_Wu)
2919     elif seleccion_1 == "Viga empotrada":
2920         dibujar_viga_empotrada_1(canvas_viga_analisis, valor_carga, valor_Wu)
2921
2922 # Inicializar Wu_var globalmente para que no se cree nuevamente al abrir la
ventana
2923 Wu_var = tk.StringVar()
2924
2925
2926
2927 def aplicar_cargas():
2928     global label_tipo_viga_cargas, entry_Wu_cargas, entry_luz_cargas,
entry_WD_cargas, entry_WL_cargas, label_titulo_viga, label_titulo_cargas,
combo_seccion_analisis

```

```
2929     global label_texto_cargas, label_reaccion
2930     # Verificar que los elementos existan antes de usarlos
2931     if label_tipo_viga_cargas is None or entry_Wu_cargas is None or
entry_luz_cargas is None or entry_WD_cargas is None or entry_WL_cargas is None:
2932         return
2933
2934     # Obtener el tipo de viga desde el label
2935     tipo_viga = label_tipo_viga_cargas.cget("text") # Obtener el texto del label
2936     if not tipo_viga: # Si el texto está vacío, no hacer nada
2937         return
2938
2939     # Obtener la sección seleccionada del combobox
2940     seccion_viga = combo_seccion_analisis.get()
2941     if not seccion_viga: # Si no se selecciona una sección, no hacer nada
2942         return
2943
2944     def animar_texto():
2945
2946         # Obtiene el texto actual
2947         texto = label_texto_cargas.cget("text")
2948
2949         # Obtiene el tamaño de la ventana y la posición actual del texto
2950         x_actual = label_texto_cargas.winfo_x()
2951
2952         # Definir el límite del movimiento (puedes cambiar estos valores)
2953         limite_derecha = 380
2954         limite_izquierda = -len(texto) * 7
2955
2956         # Mover el texto
2957         if x_actual < limite_derecha:
2958             label_texto_cargas.place(x=x_actual + 1, y=3)
2959             label_texto_cargas.after(50, animar_texto)
2960         else:
2961             # Si llega al final, vuelve a la posición inicial
2962             label_texto_cargas.place(x=limite_izquierda, y=3)
2963             label_texto_cargas.after(50, animar_texto)
2964
2965         # Iniciar la animación
2966         animar_texto()
2967
2968     try:
2969         # Longitud de la viga en unidades
2970         longitud = 6 * 100 # Longitud fija de 6 metros (600 unidades)
2971
2972         # Obtener Wu como flotante
2973         try:
2974             carga_distribuida = float(entry_Wu_cargas.get())
2975         except ValueError:
2976             carga_distribuida = 0 # Si no es válido, usar 0 como carga
2977
2978         # Obtener el valor de la luz de la viga desde entry_luz_cargas
2979         try:
2980             luz_viga = float(entry_luz_cargas.get())
2981         except ValueError:
2982             luz_viga = 0 # Si no es válido, usar 0 como valor de luz
2983
2984         # Obtener los valores de WD y WL
2985         try:
2986             WD = float(entry_WD_cargas.get())
2987         except ValueError:
```

```

2988     WD = 0 # Si no es válido, usar 0 como valor de WD
2989
2990     try:
2991         WL = float(entry_WL_cargas.get())
2992     except ValueError:
2993         WL = 0 # Si no es válido, usar 0 como valor de WL
2994
2995
2996     # Inicializar momento_A y momento_B como 0 por defecto
2997     momento_A = 0
2998     momento_B = 0
2999
3000
3001     # Calcular las reacciones
3002     if tipo_viga == "Viga apoyada":
3003         # Reacciones en los apoyos para viga apoyada
3004         reaccion_apoyo = carga_distribuida * luz_viga / 2 #  $R = (W_u * L) / 2$ 
3005         texto_reacciones = [f"Ray = {reaccion_apoyo:.2f} kN", f"Rby =
{reaccion_apoyo:.2f} kN"]
3006         # Actualizar el label de reacciones con el valor de reaccion_apoyo
3007         label_reaccion.config(text=f"{reaccion_apoyo:.2f} kN")
3008         # Actualizar el título de la viga
3009         label_titulo_viga.config(text="Análisis de Viga apoyada")
3010     elif tipo_viga == "Viga empotrada":
3011         # Reacciones en los apoyos para viga empotrada
3012         reaccion_apoyo = carga_distribuida * luz_viga / 2 # Similar a viga
apoyada para este ejemplo
3013         texto_reacciones = [f"Ray = {reaccion_apoyo:.2f} kN", f"Rby =
{reaccion_apoyo:.2f} kN"]
3014         # Reacciones de momentos en los apoyos para viga empotrada
3015         momento_A = carga_distribuida * luz_viga**2 / 12 #  $MA = W_u * L^2 / 12$ 
3016         momento_B = -carga_distribuida * luz_viga**2 / 12 #  $MB = -W_u * L^2 /
12$ 
3017         texto_reacciones = [f"Ray = {reaccion_apoyo:.2f} kN", f"Rby =
{reaccion_apoyo:.2f} kN"]
3018         texto_momentos = [f"MA = {momento_A:.2f} kN·m", f"MB = {momento_B:.2f}
kN·m"]
3019         # Actualizar el label de reacciones con el valor de reaccion_apoyo
3020         label_reaccion.config(text=f"{reaccion_apoyo:.2f} kN")
3021         # Actualizar el título de la viga
3022         label_titulo_viga.config(text="Análisis de Viga empotrada")
3023     else:
3024         texto_reacciones = ["Reacciones no calculadas"]
3025
3026     # Aquí es donde actualizamos el texto de `label_texto_cargas`
3027     mensaje_analisis = f"!Muy bien! El análisis de la viga con sección
{seccion_viga} es correcto."
3028     label_texto_cargas.config(text=mensaje_analisis) # Actualizar el texto
del label_texto_cargas
3029
3030     # Crear figura y eje para el dibujo
3031     fig = Figure(figsize=(6, 4), dpi=100)
3032     ax = fig.add_subplot(111)
3033
3034     # Configuración general
3035     desplazamiento_x = 10
3036     eje_y_central = 0
3037     altura_triangulo = 40
3038     altura_carga = 60
3039

```

```

3040 # Dibujar la viga
3041 ax.plot([desplazamiento_x, longitud + desplazamiento_x],
3042         [eje_y_central, eje_y_central],
3043         color='black', linewidth=3, label="Viga")
3044
3045 # Dibujar apoyos según el tipo de viga
3046 if tipo_viga == "Viga apoyada":
3047     # Apoyo inicial (rellenar triángulo con color azul y borde negro)
3048     ax.fill([desplazamiento_x - 20, desplazamiento_x, desplazamiento_x +
3049 20],
3049             [eje_y_central - altura_triángulo, eje_y_central,
3050 eje_y_central - altura_triángulo],
3050             color='blue', edgecolor='black', linewidth=2) # Contorno
3051 negro
3052
3053     # Apoyo final (rellenar triángulo con color azul y borde negro)
3054     ax.fill([longitud + desplazamiento_x - 20, longitud +
3055 desplazamiento_x, longitud + desplazamiento_x + 20],
3056             [eje_y_central - altura_triángulo, eje_y_central,
3057 eje_y_central - altura_triángulo],
3058             color='blue', edgecolor='black', linewidth=2) # Contorno
3059 negro
3060
3061 if tipo_viga == "Viga empotrada":
3062     # Empotramiento inicial (línea vertical)
3063     ax.plot([desplazamiento_x, desplazamiento_x],
3064             [eje_y_central - 50, eje_y_central + 50],
3065             color='black', linewidth=3)
3066
3067     # Empotramiento final (línea vertical)
3068     ax.plot([longitud + desplazamiento_x, longitud + desplazamiento_x],
3069             [eje_y_central - 50, eje_y_central + 50],
3070             color='black', linewidth=3)
3071
3072 # Dibujar la carga distribuida si Wu > 0
3073 if carga_distribuida > 0:
3074     x_inicio = desplazamiento_x
3075     x_fin = longitud + desplazamiento_x
3076     num_flechas = 10
3077     espacio_flechas = (x_fin - x_inicio) / num_flechas
3078
3079     # Dibujar fondo de la carga
3080     ax.fill_between([x_inicio, x_fin],
3081                     eje_y_central + altura_carga + 5,
3082                     eje_y_central,
3083                     color='lime', alpha=0.5)
3084
3085     # Dibujar flechas de la carga
3086     for i in range(num_flechas + 1):
3087         x_pos = x_inicio + i * espacio_flechas
3088         ax.arrow(x_pos, eje_y_central + altura_carga, 0, -40,
3089                 head_width=10, head_length=15, fc='blue', ec='blue')
3090
3091     # Dibujar línea superior de la carga
3092     ax.plot([x_inicio, x_fin],
3093             [eje_y_central + altura_carga, eje_y_central + altura_carga],
3094             color='blue', linewidth=2, label=f"Wu={carga_distribuida}")
3095
3096     # Agregar el texto de la carga distribuida en el gráfico
3097     texto_carga = f"Wu = {carga_distribuida} kN/m"

```

```

3093         ax.text(longitud / 2 + desplazamiento_x, eje_y_central + altura_carga
+ 30,
3094                 texto_carga, ha='center', va='center', fontsize=8,
color='blue')
3095
3096         # Mostrar el texto de la cota debajo de la viga
3097         texto_cota = f"Luz = {luz_viga} m"
3098         ax.text(longitud / 2 + desplazamiento_x, eje_y_central - 40,
3099                 texto_cota, ha='center', va='center', fontsize=8, color='black')
3100
3101         # Crear marcadores personalizados para la leyenda
3102         handles_reacciones = [
3103             Line2D([0], [0], linestyle='None', color='none', label="Reacciones:"),
3104             # Subtítulo como línea vacía
3105             Line2D([0], [0], marker='^', color='black', markerfacecolor='blue',
linestyle='None', markersize=8),
3106             Line2D([0], [0], marker='^', color='black', markerfacecolor='blue',
linestyle='None', markersize=8)
3107         ]
3108         handles_reacciones_M = [
3109             Line2D([0], [0], linestyle='None', color='none', label="Reacciones:"),
3110             # Subtítulo como línea vacía
3111             Line2D([0], [0], marker='^', color='black', markerfacecolor='black',
linestyle='None', markersize=8),
3112             Line2D([0], [0], marker='^', color='black', markerfacecolor='black',
linestyle='None', markersize=8)
3113         ]
3114         handles_momentos = [
3115             Line2D([0], [0], linestyle='None', color='none', label="Momentos:"),
3116             # Subtítulo como línea vacía
3117             Line2D([0], [0], marker='o', color='black', markerfacecolor='red',
linestyle='None', markersize=8),
3118             Line2D([0], [0], marker='o', color='black', markerfacecolor='red',
linestyle='None', markersize=8)
3119         ]
3120
3121         # Crear leyendas para las cargas WD y WL
3122         handles_WD = [
3123             Line2D([0], [0], linestyle='None', color='none', label="Carga
muerta:"),
3124             Line2D([0], [0], marker='s', color='black', markerfacecolor='lime',
linestyle='None', markersize=8)
3125         ]
3126         handles_WL = [
3127             Line2D([0], [0], linestyle='None', color='none', label="Carga viva:"),
3128             Line2D([0], [0], marker='s', color='black', markerfacecolor='lime',
linestyle='None', markersize=8)
3129         ]
3130
3131         # Leyendas para la viga apoyada
3132         if tipo_viga == "Viga apoyada":
3133             ax.legend(
3134                 handles=handles_reacciones,
3135                 labels=["Reacciones:", # Subtítulo
f"Ray = {reaccion_apoyo:.2f} kN", # Reacción en A
f"Rby = {reaccion_apoyo:.2f} kN"], # Reacción en B
3136                 loc='upper left',
3137                 bbox_to_anchor=(0.5, 1.8),
3138                 fontsize=8,
3139                 frameon=True,

```

```

3139         edgecolor='black',
3140         facecolor='lightgray'
3141     )
3142     fig.legend(
3143         handles=handles_WD + handles_WL,
3144         labels=["Carga muerta:", # Subtítulo
3145                 f"WD = {WD:.2f} kN/m", # WD
3146                 "Carga viva:", # Subtítulo
3147                 f"WL = {WL:.2f} kN/m"], # WL
3148         loc='upper left',
3149         bbox_to_anchor=(0.1, 0.98), # Ajusta la posición
3150         fontsize=8,
3151         frameon=True,
3152         edgecolor='black',
3153         facecolor='lightgray'
3154     )
3155
3156     # Leyendas para la viga empotrada
3157     elif tipo_viga == "Viga empotrada":
3158         # Leyenda para reacciones
3159         fig.legend(
3160             handles=handles_reacciones_M,
3161             labels=["Reacciones:", # Subtítulo
3162                     f"Ray = {reaccion_apoyo:.2f} kN", # Reacción en A
3163                     f"Rby = {reaccion_apoyo:.2f} kN"], # Reacción en B
3164             loc='upper left',
3165             bbox_to_anchor=(0.55, 0.98), # Ajusta la posición
3166             fontsize=8,
3167             frameon=True,
3168             edgecolor='black',
3169             facecolor='lightgray'
3170         )
3171
3172         # Leyenda para momentos
3173         fig.legend(
3174             handles=handles_momentos,
3175             labels=["Momentos:", # Subtítulo
3176                     f"MA = {momento_A:.2f} kNm", # Momento en A
3177                     f"MB = {momento_B:.2f} kNm"], # Momento en B
3178             loc='upper left',
3179             bbox_to_anchor=(0.1, 0.98), # Ajusta la posición debajo de la
3180             # leyenda de reacciones
3181             fontsize=8,
3182             frameon=True,
3183             edgecolor='black',
3184             facecolor='lightgray'
3185         )
3186
3187         # Añadir leyenda de cargas WD y WL al gráfico
3188         fig.legend(
3189             handles=handles_WD,
3190             labels=["Carga muerta:", f"WD = {WD:.2f} kN/m"],
3191             loc='upper left',
3192             bbox_to_anchor=(0.55, 0.3), # Ajustar la posición de la leyenda
3193             fontsize=8,
3194             frameon=True,
3195             edgecolor='black',
3196             facecolor='lightgray'
3197         )
3198         fig.legend(

```

```

3198         handles=handles_WL,
3199         labels=["Carga viva:",f"WL = {WL:.2f} kN/m"],
3200         loc='upper left',
3201         bbox_to_anchor=(0.1, 0.3), # Ajustar la posición de la leyenda
3202         fontsize=8,
3203         frameon=True,
3204         edgecolor='black',
3205         facecolor='lightgray'
3206     )
3207
3208     # Ajustar límites y limpiar canvas
3209     ax.set_xlim(desplazamiento_x - 50, longitud + desplazamiento_x + 50)
3210     ax.set_ylim(eje_y_central - 100, eje_y_central + 150)
3211     ax.set_aspect('equal', adjustable='box')
3212     ax.axis('off')
3213
3214     # Limpiar canvas antes de mostrar nueva figura
3215     for widget in canvas_1.wininfo_children():
3216         widget.destroy()
3217
3218     # Mostrar figura en canvas
3219     canvas_viga_cargas = FigureCanvasTkAgg(fig, master=canvas_1)
3220     canvas_viga_cargas.draw()
3221     canvas_viga_cargas.get_tk_widget().place(x=1, y=1, width=380, height=280)
3222
3223     except Exception as e:
3224         print(f"Error al dibujar la viga: {e}")
3225
3226     # Variables globales para los Label y StringVar
3227     label_seccion_luz = None
3228     label_seccion_Wu = None
3229     var_luz_cargas = None
3230     var_Wu_cargas = None
3231
3232     def abrir_ventana_cargas():
3233         global ventana_cargas_abierta , combo_seccion_analisis, entry_base_cargas,
3234         entry_altura_cargas, entry_luz_cargas
3235         global entry_Wpp_cargas, entry_WD_cargas, entry_WL_cargas,
3236         entry_Wtotal_cargas, entry_Wu_cargas
3237         global
3238         Wu_var, label_tipo_viga_cargas, combo_viga_cargas, canvas_1, label_titulo_viga, label_t
3239         itulo_cargas, label_texto_cargas
3240         global var_luz_cargas, var_Wu_cargas, label_seccion_luz,
3241         label_seccion_Wu, canvas_viga_analisis
3242         # Verificar si ya hay una ventana abierta
3243         if ventana_cargas_abierta is not None and
3244         ventana_cargas_abierta.wininfo_exists():
3245             # Si la ventana ya está abierta, no hacemos nada
3246             return
3247
3248         # Crear una nueva ventana solo si no existe
3249         ventana_cargas_abierta = tk.Toplevel() # Toplevel crea una ventana secundaria
3250
3251         # Configurar la ventana
3252         ventana_cargas_abierta.title("Análisis de cargas")
3253         ventana_cargas_abierta.geometry("560x600") # Tamaño de la ventana
3254
3255         # Evitar que la ventana sea maximizada o redimensionada
3256         ventana_cargas_abierta.resizable(False, False)

```

```

3252 # Mantener la ventana siempre encima
3253 ventana_cargas_abierta.attributes('-topmost', True)
3254
3255
3256 # Crear contenido en la ventana cargas
3257 labelframe_cargas= tk.LabelFrame(ventana_cargas_abierta,text="")
3258 labelframe_cargas.place(x=30,y=30,width=500,height=500)
3259 label_apoyo_cargas = tk.Label(labelframe_cargas,text="Tipo de apoyo:")
3260 label_apoyo_cargas.place(x=20,y=20)
3261 entry_apoyo_cargas = tk.Entry(labelframe_cargas)
3262 entry_apoyo_cargas.place(x=120,y=20,width=100)
3263 # Obtener el valor del Label 'label_restriccion_apoyo' y mostrarlo en el Entry
3264 valor_restriccion = label_restriccion_apoyo.cget("text") # Obtener el texto
del Label
3265 entry_apoyo_cargas.insert(0, valor_restriccion) # Insertamos el valor en el
Entry
3266 label_viga_cargas = tk.Label(labelframe_cargas,text="Tipo de viga:")
3267 label_viga_cargas.place(x=270,y=20)
3268 # Crear el Combobox con las opciones
3269 combo_viga_cargas = ttk.Combobox(labelframe_cargas, values=["Viga apoyada",
"Viga empotrada"])
3270 combo_viga_cargas.place(x=350, y=20, width=120)
3271 combo_viga_cargas.set("Seleccionar") # Establecer el valor inicial
3272
3273 labelframe_viga_cargas= tk.LabelFrame(labelframe_cargas,text="Diagrama viga")
3274 labelframe_viga_cargas.place(x=20,y=70,width=457,height=205)
3275 canvas_viga_analisis =
tk.Canvas(labelframe_viga_cargas,bg="white",bd=2,relief="ridge")
3276 canvas_viga_analisis.pack(fill=tk.BOTH, expand=True, padx=0, pady=0) #
Incrustar el canvas con ajuste
3277
3278 #Crear botones Aceptar, Calcular , Aplicar
3279 boton_aceptar_cargas= ttk.Button(ventana_cargas_abierta,text="Aceptar")
3280 boton_aceptar_cargas.place(x=110,y=550,width=100)
3281
3282 boton_aplicar_cargas= ttk.Button(ventana_cargas_abierta,text="Aplicar",
command=aplicar_cargas)
3283 boton_aplicar_cargas.place(x=330,y=550,width=100)
3284
3285
3286 # Crear las variables de tipo StringVar para vincular con los Entry
3287 var_luz_cargas = tk.StringVar()
3288 var_Wu_cargas = tk.StringVar()
3289
3290 labelframe_analisis = tk.LabelFrame(labelframe_cargas,text="Análisis de
cargas")
3291 labelframe_analisis.place(x=20,y=280,width=457,height=180)
3292 label_seccion_analisis =tk.Label(labelframe_analisis,text="Sección:")
3293 label_seccion_analisis.place(x=10,y=40)
3294 combo_seccion_analisis=ttk.Combobox(labelframe_analisis)
3295 combo_seccion_analisis.place(x=70,y=40,width=95)
3296 combo_seccion_analisis.set("Seleccionar")
3297 label_unidad_cargas1= tk.Label(labelframe_analisis,text="[VG]")
3298 label_unidad_cargas1.place(x=175,y=40)
3299 luz_cargas = tk.Label(labelframe_analisis,text="Luz:")
3300 luz_cargas.place(x=10,y=10)
3301 entry_luz_cargas =tk.Entry(labelframe_analisis,textvariable=var_luz_cargas)
3302 entry_luz_cargas.place(x=70,y=10,width=95)
3303 label_unidad_cargas2= tk.Label(labelframe_analisis,text="[m]")
3304 label_unidad_cargas2.place(x=175,y=10)

```

```

3305 base_cargas = tk.Label(labelframe_analisis,text="Base:")
3306 base_cargas.place(x=10,y=70)
3307 entry_base_cargas = tk.Entry(labelframe_analisis)
3308 entry_base_cargas.place(x=70,y=70,width=95)
3309 label_unidad_cargas3= tk.Label(labelframe_analisis,text="[m]")
3310 label_unidad_cargas3.place(x=175,y=70)
3311 altura_cargas = tk.Label(labelframe_analisis,text="Altura:")
3312 altura_cargas.place(x=10,y=100)
3313 entry_altura_cargas = tk.Entry(labelframe_analisis)
3314 entry_altura_cargas.place(x=70,y=100,width=95)
3315 label_unidad_cargas4= tk.Label(labelframe_analisis,text="[m]")
3316 label_unidad_cargas4.place(x=175,y=100)
3317 peso_concreto_cargas = tk.Label(labelframe_analisis,text="γConcreto:")
3318 peso_concreto_cargas.place(x=5,y=130)
3319 entry_concreto_cargas = tk.Entry(labelframe_analisis)
3320 entry_concreto_cargas.place(x=70,y=130,width=95)
3321 entry_concreto_cargas.insert(0, "24.00")
3322 label_unidad_cargas5= tk.Label(labelframe_analisis,text="[kN/m³]")
3323 label_unidad_cargas5.place(x=165,y=130)
3324 frame_cargas = tk.Frame(labelframe_analisis,bg="grey",bd=4,relief="ridge")
3325 frame_cargas.place(x=230,y=10,width=1,height=140)
3326 WD_cargas = tk.Label(labelframe_analisis,text="WD:")
3327 WD_cargas.place(x=250,y=10)
3328 entry_WD_cargas= tk.Entry(labelframe_analisis)
3329 entry_WD_cargas.place(x=290,y=10,width=95)
3330 label_unidad_WD = tk.Label(labelframe_analisis,text="[kN/m]")
3331 label_unidad_WD.place(x=390,y=10)
3332 WL_cargas = tk.Label(labelframe_analisis,text="WL:")
3333 WL_cargas.place(x=250,y=40)
3334 entry_WL_cargas= tk.Entry(labelframe_analisis)
3335 entry_WL_cargas.place(x=290,y=40,width=95)
3336 label_unidad_WL = tk.Label(labelframe_analisis,text="[kN/m]")
3337 label_unidad_WL.place(x=390,y=40)
3338 Wpp_cargas = tk.Label(labelframe_analisis,text="Wpp:")
3339 Wpp_cargas.place(x=250,y=70)
3340 entry_Wpp_cargas= tk.Entry(labelframe_analisis)
3341 entry_Wpp_cargas.place(x=290,y=70,width=95)
3342 label_unidad_Wpp = tk.Label(labelframe_analisis,text="[kN/m]")
3343 label_unidad_Wpp.place(x=390,y=70)
3344 Wtotal_cargas = tk.Label(labelframe_analisis,text="Wtotal:")
3345 Wtotal_cargas.place(x=245,y=100)
3346 entry_Wtotal_cargas= tk.Entry(labelframe_analisis)
3347 entry_Wtotal_cargas.place(x=290,y=100,width=95)
3348 label_unidad_Wtotal = tk.Label(labelframe_analisis,text="[kN/m]")
3349 label_unidad_Wtotal.place(x=390,y=100)
3350 Wu_cargas = tk.Label(labelframe_analisis,text="Wu:")
3351 Wu_cargas.place(x=250,y=130)
3352 entry_Wu_cargas= tk.Entry(labelframe_analisis,textvariable=var_Wu_cargas)
3353 entry_Wu_cargas.place(x=290,y=130,width=95)
3354 label_unidad_Wu = tk.Label(labelframe_analisis,text="[kN/m]")
3355 label_unidad_Wu.place(x=390,y=130)
3356
3357 # Vincular eventos
3358 combo_viga_cargas.bind("<<ComboboxSelected>>", lambda event:
actualizar_diagrama_viga(event, combo_viga_cargas, canvas_viga_analisis,
entry_luz_cargas,Wu_var,label_tipo_viga_cargas))
3359 entry_luz_cargas.bind("<KeyRelease>", lambda event:
actualizar_diagrama_viga(event, combo_viga_cargas, canvas_viga_analisis,
entry_luz_cargas,Wu_var,label_tipo_viga_cargas))

```

```

3360 # Llamamos a la función para actualizar el ComboBox con el valor de
label_seccion_cargas
3361 actualizar_combo_seccion(combo_seccion_analisis)
3362
3363 # Asociar el evento de selección del ComboBox a la función actualizar_entries
3364 combo_seccion_analisis.bind("<<ComboboxSelected>>", actualizar_entries)
3365 # Asociar eventos para actualizar automáticamente Wtotal
3366 entry_WD_cargas.bind("<KeyRelease>", calcular_Wtotal)
3367 entry_WL_cargas.bind("<KeyRelease>", calcular_Wtotal)
3368
3369 # Vincular el trace solo una vez cuando se cree la ventana
3370 Wu_var.trace_add("write", lambda name, index, mode:
actualizar_diagrama_viga(None, combo_viga_cargas, canvas_viga_analisis,
entry_luz_cargas, Wu_var, label_tipo_viga_cargas))
3371
3372
3373 # Función para actualizar los labels con los valores de los Entry
3374 def actualizar_labels(*args):
3375     label_seccion_luz.config(text=var_luz_cargas.get()) # Actualiza el label
con el valor de luz de cargas
3376     label_seccion_Wu.config(text=var_Wu_cargas.get()) # Actualiza el label
con el valor de Wu de cargas
3377
3378 # Vincular las variables de los Entry con la función de actualización
3379 var_luz_cargas.trace("w", actualizar_labels) # Cuando cambia la variable,
actualiza el label correspondiente
3380 var_Wu_cargas.trace("w", actualizar_labels) # Lo mismo para Wu de Cargas
3381
3382 # Función para cerrar la ventana y liberar la referencia
3383 def cerrar_ventana():
3384     global ventana_cargas_abierta
3385     ventana_cargas_abierta.destroy()
3386     ventana_cargas_abierta = None
3387
3388
3389 label_seccion_luz = tk.Label(ventana, text="", bg="red")
3390 #label_seccion_luz.place(x=500, y=450, width=100) #<---- Dato que hare salto
de codigo luz de viga
3391
3392 label_seccion_Wu = tk.Label(ventana, text="", bg="red") #<---- Dato que hare salto
de codigo Wu
3393 #label_seccion_Wu.place(x=500, y=480, width=100)
3394
3395 label_tipo_viga_cargas = tk.Label(ventana, text="", bg="red")
3396 #label_tipo_viga_cargas.place(x=500, y=520, width=100) #<--- Dato que hare
salto de codigo tipo viga apoyada o viga empotrada
3397
3398 label_restriccion_apoyo= tk.Label(ventana, text="", bg="red")
3399 #label_restriccion_apoyo.place(x=500, y=400, width=100) #<---- Dato que hare salto
de codigo condicion de apoyos
3400
3401 label_seccion_cargas = tk.Label(ventana, text="", bg="red")
3402 #label_seccion_cargas.place(x=500, y=420, width=100) #<----- Dato que hare
salto de codigo nombre de seccion transversal
3403
3404 frame_ventana_seccion = tk.Frame(ventana, bd=10, relief="ridge", bg="white")
3405 frame_ventana_seccion.place(x=36, y=405, width=400, height=370)
3406
3407 frame_ventana_falsa =
tk.Frame(frame_ventana_seccion, bd=2, relief="flat", bg="white")

```

```

3408 frame_ventana_falsa.place(x=1,y=35,width=378,height=280)
3409
3410 label_titulo_seccion = tk.Label(frame_ventana_seccion ,text="Diagrama sección
transversal",bd=2,relief="raise")
3411 label_titulo_seccion.place(x=1,y=1,width=380,height=35)
3412
3413 label_titulo_falsa = tk.Frame(frame_ventana_seccion,bd=2,relief="raise")
3414 label_titulo_falsa.place(x=1,y=315,width=380,height=35)
3415
3416 frame_ventana_viga= tk.Frame(ventana, bd=10, relief="ridge",bg="white")
3417 frame_ventana_viga.place(x=437, y=35, width=400, height=370)
3418
3419 label_titulo_viga = tk.Label(frame_ventana_viga,text="Análisis de Viga", bd=2,
relief="raise")
3420 label_titulo_viga.place(x=1,y=1,width=380,height=35)
3421
3422 label_titulo_cargas = tk.Frame(frame_ventana_viga,bd=2,relief="raise")
3423 label_titulo_cargas.place(x=1,y=315,width=380,height=35)
3424
3425 label_texto_cargas = tk.Label(label_titulo_cargas, text="")
3426 label_texto_cargas.pack(fill=tk.BOTH, expand=True)
3427
3428 canvas_1 = tk.Canvas(frame_ventana_viga, bd=2, relief="flat", bg="white")
3429 canvas_1.place(x=1, y=36, width=380, height=280)
3430
3431 def actualizar_contenido_graficos(label_seccion_cargas, label_seccion_luz,
label_seccion_Wu, label_tipo_viga_cargas, label_graficos_viga, label_graficos_luz,
label_graficos_Wu, combo_graficos_Wu, label_graficos_tipo_viga):
3432     """
3433     Función para actualizar varios widgets en la ventana de gráficos, pasando como
parámetros
3434     los widgets necesarios.
3435     """
3436     # Tomamos el texto de label_seccion_cargas de la ventana principal
3437     valor_cargas = label_seccion_cargas.cget("text")
3438
3439     # Actualizamos el texto del label_graficos_viga con el valor de
label_seccion_cargas
3440     label_graficos_viga.config(text=valor_cargas)
3441
3442     # Actualizar el texto del label_graficos_luz con el valor de label_seccion_luz
3443     valor_luz = label_seccion_luz.cget("text")
3444     valor_luz_formateado = f"{valor_luz} m"
3445     label_graficos_luz.config(text=valor_luz_formateado)
3446
3447     # Agregar el valor de label_seccion_Wu como opción al combo
3448     valor_Wu_graficos = label_seccion_Wu.cget("text")
3449
3450     # Añadir el valor real del Wu al ComboBox, sin seleccionarlo por defecto
3451     combo_graficos_Wu['values'] = ["Seleccionar", valor_Wu_graficos]
3452     combo_graficos_Wu.set("Seleccionar")
3453
3454     # Tomamos el valor de label_tipo_viga_cargas y actualizamos
label_graficos_tipo_viga
3455     valor_tipo_viga = label_tipo_viga_cargas.cget("text")
3456     label_graficos_tipo_viga.config(text=valor_tipo_viga)
3457
3458 # Función para mostrar la ventana emergente
3459 def mostrar_ventana_emergente():

```

```
3460     messagebox.showinfo("Pilot Structural", "¡Muy bien! Ahora puedes iniciar con  
las configuraciones de diseño a flexión.")  
3461  
3462 # Función para cerrar la ventana de gráficos  
3463 def cerrar_ventana_graficos():  
3464     global ventana_graficos_abierta, ventana_graficos  
3465     # Mostrar la ventana emergente antes de cerrar  
3466     mostrar_ventana_emergente()  
3467  
3468     ventana_graficos.destroy() # Cerrar la ventana de gráficos  
3469     ventana_graficos_abierta = False # Permitir reabrir la ventana  
3470  
3471 # Definir las variables globales  
3472 ventana_graficos_abierta = False  
3473 ventana_graficos = None  
3474  
3475 # Variables globales para los labels y combobox  
3476 label_valor_M1 = None  
3477 label_valor_M2 = None  
3478 label_valor_M3 = None  
3479 Momento_Mu = None  
3480 label_valor_Mmax = None  
3481  
3482 # Función para abrir la ventana de gráficos  
3483 def abrir_ventana_graficos():  
3484     global ventana_graficos_abierta, ventana_graficos, combo_graficos_Wu,  
label_graficos_luz, label_graficos_tipo_viga, label_frame_graficos_V_M  
3485     global datos_exportar, boton_cancelar_graficos  
3486     global fig  
3487     global label_valor_M1, label_valor_M2, label_valor_M3, label_valor_Mmax  
3488     # Verificar si la ventana ya está abierta  
3489     if ventana_graficos_abierta:  
3490         return # Si ya está abierta, no hacer nada  
3491  
3492     # Si no está abierta, crear la ventana secundaria  
3493     ventana_graficos = tk.Toplevel()  
3494     ventana_graficos.title("Diagramas cortante y momento")  
3495     ventana_graficos.geometry("600x620")  
3496  
3497     # Configuración adicional: no redimensionable y siempre encima de otras  
ventanas  
3498     ventana_graficos.resizable(False, False)  
3499     ventana_graficos.attributes('-topmost', True)  
3500     # Marcar la ventana como abierta  
3501     ventana_graficos_abierta = True  
3502  
3503     # Crear contenido en la ventana graficos  
3504  
3505     label_frame_graficos = tk.LabelFrame(ventana_graficos, text="")  
3506     label_frame_graficos.place(x=20, y=20, width=560, height=550)  
3507     label_frame_carga = tk.LabelFrame(label_frame_graficos, text="Condiciones de  
carga [Wu]")  
3508     label_frame_carga.place(x=10, y=10, width=200, height=100)  
3509     label_graficos_Wu = tk.Label(label_frame_carga, text="Wu:")  
3510     label_graficos_Wu.place(x=10, y=40)  
3511     combo_graficos_Wu = ttk.Combobox(label_frame_carga)  
3512     combo_graficos_Wu.place(x=40, y=40, width=130)  
3513     combo_graficos_Wu.set("Seleccionar")  
3514     label_graficos_info = tk.Label(label_frame_carga, text="Carga  
mayorada", bd=1, relief="sunken")
```

```

3515     label_graficos_info.place(x=10,y=10,width=160)
3516     label_frame_graficos_seccion =tk.LabelFrame(label_frame_graficos,text="Sección
[VG]")
3517     label_frame_graficos_seccion.place(x=220,y=10,width=150,height=100)
3518     label_graficos_viga =
tk.Label(label_frame_graficos_seccion,text="",bd=1,relief="sunken")
3519     label_graficos_viga.place(x=10,y=10,width=120)
3520     label_luz_viga = tk.Label(label_frame_graficos_seccion,text="Luz:")
3521     label_luz_viga.place(x=10,y=40)
3522     label_graficos_luz
=tk.Label(label_frame_graficos_seccion,text="",bd=1,relief="sunken")
3523     label_graficos_luz.place(x=50,y=40,width=80)
3524     label_graficos_tipo_viga = tk.Label(label_frame_graficos,text="",bg="red")
3525     #label_graficos_tipo_viga.place(x=10,y=80,width=100)
3526     label_frame_graficos_slider = tk.LabelFrame(label_frame_graficos,text="Ajuste
de valor")
3527     label_frame_graficos_slider.place(x=380,y=10,width=165,height=100)
3528     # Crear un slider
3529     slider_valor = tk.Scale(label_frame_graficos_slider,
orient="horizontal",from_=0, to=10,resolution=0.1)
3530     slider_valor.place(x=25,y=0)
3531     label_graficos_cortante =
tk.Label(label_frame_graficos_slider,text="Cortante:")
3532     label_graficos_cortante.place(x=10,y=40)
3533     label_graficos_V = tk.Label(label_frame_graficos_slider,text="")
3534     label_graficos_V.place(x=70,y=40,width=75)
3535     label_graficos_momento = tk.Label(label_frame_graficos_slider,text="Momento:")
3536     label_graficos_momento.place(x=7,y=60)
3537     label_graficos_M = tk.Label(label_frame_graficos_slider,text="")
3538     label_graficos_M.place(x=70,y=60,width=75)
3539
3540     #Crear labelframe valores max y min Cortante
3541     label_frame_Valores_V = tk.LabelFrame(label_frame_graficos,text="Valores max y
min [V]")
3542     label_frame_Valores_V.place(x=380,y=120,width=165,height=100)
3543     label_valor_max = tk.Label(label_frame_Valores_V,text="V[+]:")
3544     label_valor_max.place(x=10,y=10)
3545     label_valor_min = tk.Label(label_frame_Valores_V,text="V[-]:")
3546     label_valor_min.place(x=10,y=45)
3547     label_max= tk.Label(label_frame_Valores_V,text="")
3548     label_max.place(x=50,y=10,width=75)
3549     label_min= tk.Label(label_frame_Valores_V,text="")
3550     label_min.place(x=50,y=45,width=75)
3551
3552     #Crear labelframe valores max y min Momento viga empotrada
3553     label_frame_Valores_M = tk.LabelFrame(label_frame_graficos,text="Valores max y
min [M]")
3554     label_frame_Valores_M.place(x=380,y=230,width=165,height=100)
3555     label_valor_min_M1 = tk.Label(label_frame_Valores_M ,text="M[-]:")
3556     label_valor_min_M1.place(x=10,y=5)
3557     label_valor_max_M2= tk.Label(label_frame_Valores_M ,text="M[+]:")
3558     label_valor_max_M2.place(x=10,y=30)
3559     label_valor_min_M3= tk.Label(label_frame_Valores_M ,text="M[-]:")
3560     label_valor_min_M3.place(x=10,y=55)
3561     label_valor_M1=tk.Label(label_frame_Valores_M,text="")
3562     label_valor_M1.place(x=50,y=5,width=75)
3563     label_valor_M2=tk.Label(label_frame_Valores_M,text="")
3564     label_valor_M2.place(x=50,y=30,width=75)
3565     label_valor_M3=tk.Label(label_frame_Valores_M,text="")
3566     label_valor_M3.place(x=50,y=55,width=75)

```

```

3567
3568     #Crear labelFrame valor Mmax viga simplemente apoyada
3569     label_frame_Mmax= tk.LabelFrame(label_frame_graficos,text="")
3570     label_frame_Mmax.place(x=380,y=340,width=165,height=60)
3571     label_intervalo = tk.Label(label_frame_Mmax,text="")
3572     label_intervalo.place(x=50,y=5,width=85)
3573     label_Mmax = tk.Label(label_frame_Mmax,text="Mmax:")
3574     label_Mmax.place(x=5,yy=30)
3575     label_valor_Mmax=tk.Label(label_frame_Mmax,text="")
3576     label_valor_Mmax.place(x=50,y=30,width=75)
3577
3578     #Crear contenido Graficos V y M
3579     label_frame_graficos_V_M = tk.LabelFrame(label_frame_graficos,text="Diagrama
Fuerza Cortante y Momento Flector [V,M]")
3580     label_frame_graficos_V_M.place(x=10,y=120,width=360,height=400)
3581
3582     #Crear contendio y botones.
3583     label_frame_datos =
tk.LabelFrame(label_frame_graficos,text="Exportar/Generar")
3584     label_frame_datos.place(x=380,y=410,width=165,height=110)
3585
3586     boton_generar_pdf = ttk.Button(label_frame_datos,text="Generar")
3587     boton_generar_pdf.place(x=30,y=50,width=100)
3588
3589     boton_exportar_datos =
ttk.Button(label_frame_datos,text="Exportar",command=exportar_datos_a_excel)
3590     boton_exportar_datos.place(x=30,y=10,width=100)
3591
3592     boton_aceptar_graficos =
ttk.Button(ventana_graficos,text="Aceptar",command=copiar_imagen_graficos)
3593     boton_aceptar_graficos.place(x=150,y=580,width=100)
3594
3595     boton_cancelar_graficos =
ttk.Button(ventana_graficos,text="Cancelar",command=cerrar_ventana_graficos)
3596     boton_cancelar_graficos.place(x=330,y=580,width=100)
3597
3598     # Asociar la acción de cerrar con el protocolo WM_DELETE_WINDOW
3599     ventana_graficos.protocol("WM_DELETE_WINDOW", cerrar_ventana_graficos)
3600
3601     # Llamar a la función de actualización, pasando los labels como parámetros
3602     actualizar_contenido_graficos(
3603         label_seccion_cargas, label_seccion_luz, label_seccion_Wu,
3604         label_tipo_viga_cargas,
3605         label_graficos_viga, label_graficos_luz, label_graficos_Wu,
3606         combo_graficos_Wu,
3607         label_graficos_tipo_viga
3608     )
3609
3610     boton_cancelar_graficos=None
3611     def mostrar_grafico(event):
3612         global datos_exportar,boton_cancelar_graficos
3613         global fig
3614         tipo_viga = label_graficos_tipo_viga.cget("text")
3615         Wu = combo_graficos_Wu.get()
3616         Luz = label_graficos_luz.cget("text")
3617
3618         try:
3619             Wu = float(Wu)
3620             Luz = float(Luz.split()[0])
3621         except (ValueError, AttributeError):

```

```

3620         return
3621
3622     # Reacciones en los apoyos
3623     Apoyo_A = Wu * Luz / 2
3624     Apoyo_B = Wu * Luz / 2
3625
3626     # Actualizar los Labels con los valores estáticos
3627     label_max.config(text=f"{Apoyo_A:.2f} kN")
3628     label_min.config(text=f"{Apoyo_B:.2f} kN")
3629
3630     # Crear listas para cortante
3631     posiciones_cortante = [0, Luz]
3632     cortante = [Apoyo_A, -Apoyo_B] # Línea diagonal
3633
3634     # Crear puntos intermedios para la curva parabólica
3635     x_momento = np.linspace(0, Luz, 100) # 100 puntos para suavizar la curva
3636     momento = []
3637
3638     if tipo_viga == "Viga apoyada":
3639         # Cortante: Línea diagonal desde Apoyo_A hacia -Apoyo_B
3640         cortante = [Apoyo_A, 0, -Apoyo_B]
3641         posiciones_cortante = [0, Luz / 2, Luz]
3642
3643         # Momento flector: Curva parabólica
3644         momento = Wu * x_momento * (Luz - x_momento) / 2
3645
3646         # Actualizar labels específicos para viga apoyada
3647         label_valor_Mmax.config(text=f"{max(momento):.2f} kNm")
3648         label_intervalo.config(text=f"[0.0 ≤ X ≤ {Luz:.2f}]")
3649
3650         label_valor_M1.config(text="N/A")
3651         label_valor_M2.config(text="N/A")
3652         label_valor_M3.config(text="N/A")
3653
3654         # Actualizar el combobox con los valores de Mmax
3655         actualizar_combobox("Viga apoyada")
3656
3657     elif tipo_viga == "Viga empotrada":
3658         # Cortante: Línea diagonal desde Apoyo_A hacia -Apoyo_B
3659         cortante = [Apoyo_A, 0, -Apoyo_B]
3660         posiciones_cortante = [0, Luz / 2, Luz]
3661
3662         # Momento flector: Curva parabólica con momentos en los extremos
3663         MA = Wu * (Luz ** 2) / 12 # Momento en apoyo izquierdo
3664         MB = -Wu * (Luz ** 2) / 12 # Momento en apoyo derecho
3665
3666         # Fórmula general del momento en viga empotrada
3667         momento = -(MB + Apoyo_B * x_momento - (Wu * x_momento ** 2) / 2)
3668
3669         label_valor_M1.config(text=f"{MA:.2f} kNm")
3670         label_valor_M2.config(text=f"{max(momento)/2:.2f} kNm")
3671         label_valor_M3.config(text=f"{MB:.2f} kNm")
3672
3673         label_valor_Mmax.config(text="N/A")
3674         label_intervalo.config(text="N/A")
3675
3676         # Actualizar el combobox con los valores de M1, M2, M3
3677         actualizar_combobox("Viga empotrada")
3678
3679         # Actualizar label_Mmax_1 con el valor de M1

```

```

3680     label_Mmax_1.config(text=f"{MA:.2f} kNm")
3681
3682     # Actualizar label_Mmax_2 con el valor de M2
3683     label_Mmax_2.config(text=f"{max(momento)/2:.2f} kNm")
3684
3685
3686     # Guardar datos en una lista global para exportar
3687     datos_exportar = {
3688         "Longitud": x_momento,
3689         "Cortante": [cortante[0] if x <= Luz / 2 else cortante[2] for x in
x_momento],
3690         "Momento": momento
3691     }
3692
3693
3694     # Crear el gráfico en el LabelFrame
3695     fig, axs = plt.subplots(2, 1, figsize=(5, 6), dpi=100, sharex=True)
3696     fig.subplots_adjust(hspace=0.3)
3697
3698     # Diagrama de Fuerza Cortante
3699     axs[0].fill_between(posiciones_cortante, 0, cortante,
where=np.array(cortante) >= 0,
3700                       interpolate=True, color="Darkblue", alpha=1)
3701     axs[0].fill_between(posiciones_cortante, 0, cortante,
where=np.array(cortante) < 0,
3702                       interpolate=True, color="red", alpha=1)
3703     axs[0].plot(posiciones_cortante, cortante, label="Fuerza Cortante",
color="blue", linestyle="-")
3704     axs[0].axhline(0, color="black", linewidth=0.8, linestyle="-")
3705     axs[0].set_title("Diagrama de Fuerza Cortante", fontsize=10)
3706     axs[0].set_xlabel("Longitud (m)", fontsize=9,)
3707     axs[0].set_ylabel("Cortante (kN)", fontsize=9)
3708     axs[0].tick_params(axis="both", labels=9)
3709
3710     # Personalización de la leyenda del Cortante
3711     axs[0].legend(loc='upper right', fontsize=9, edgecolor='black',
facecolor='white', bbox_to_anchor=(0.95, 0.95), framealpha=0.5)
3712
3713     axs[0].set_facecolor('lightgray')
3714     axs[0].set_xticks([])
3715     axs[0].set_yticks([])
3716
3717     # Diagrama de Momento Flector
3718     axs[1].fill_between(x_momento, 0, momento, where=momento >= 0,
3719                       interpolate=True, color="Darkblue", alpha=1)
3720     axs[1].fill_between(x_momento, 0, momento, where=momento < 0,
3721                       interpolate=True, color="red", alpha=1)
3722     axs[1].plot(x_momento, momento, label="Momento Flector", color="black",
linestyle="-")
3723     axs[1].axhline(0, color="black", linewidth=0.8, linestyle="-") # Línea
horizontal en 0
3724     axs[1].set_title("Diagrama de Momento Flector", fontsize=10)
3725     axs[1].set_xlabel("Longitud (m)", fontsize=9)
3726     axs[1].set_ylabel("Momento (kNm)", fontsize=9)
3727     axs[1].tick_params(axis="both", labels=9)
3728
3729     # Personalización de la leyenda del Momento Flector
3730     axs[1].legend(loc='center', fontsize=9, bbox_to_anchor=(0.5, 0.8),
edgecolor='black', facecolor='white', framealpha=0.5)
3731

```

```

3732
3733     axs[1].set_facecolor('lightgray')
3734     axs[1].set_xticks([])
3735     axs[1].set_yticks([])
3736
3737     # Fijar los límites del eje Y para mantener el 0 fijo en la intersección
3738     momento_min = min( momento)
3739     momento_max = max( momento)
3740     axs[1].set_ylim(momento_min - abs(momento_min * 0.1), momento_max +
abs(momento_max * 0.1))
3741
3742
3743
3744     # Limpiar contenido previo y mostrar el gráfico en el LabelFrame
3745     for widget in label_frame_graficos_V_M.wininfo_children():
3746         widget.destroy()
3747
3748     canvas = FigureCanvasTkAgg(fig, master=label_frame_graficos_V_M)
3749     canvas.draw()
3750     canvas.get_tk_widget().pack(fill="both", expand=True)
3751
3752     # Configurar el rango del slider basado en la longitud de la viga
3753     slider_valor.config(from_=0, to=Luz, resolution=0.1)
3754     slider_valor.set(0)
3755
3756     # Función para actualizar los marcadores y contenido widgets
3757     def actualizar_marcadores(valor):
3758         posicion = float(valor)
3759
3760         # Eliminar marcadores previos
3761         for line in axs[0].lines[1:]:
3762             line.remove()
3763         for line in axs[1].lines[1:]:
3764             line.remove()
3765
3766         # Interpolación para obtener los valores de cortante y momento en la
posición actual
3767         cortante_y = np.interp(posicion, posiciones_cortante, cortante)
3768         momento_y = np.interp(posicion, x_momento, momento)
3769
3770         cortante_y = np.interp(posicion, posiciones_cortante, cortante)
3771         axs[0].plot(posicion, cortante_y, marker="o", color="black",
linestyle=' ', label=f"[V]: {cortante_y:.2f} kN")
3772
3773         momento_y = np.interp(posicion, x_momento, momento)
3774         axs[1].plot(posicion, momento_y, marker="o", color="black",linestyle='
', label=f"[M]: {-momento_y:.2f} kNm",)
3775
3776         # Actualizar el texto del eje X con la longitud dinámica basada en la
posición del slider
3777         axs[0].set_xlabel(f"Longitud ({posicion:.2f} m)", fontsize=9)
3778         axs[1].set_xlabel(f"Longitud ({posicion:.2f} m)", fontsize=9)
3779
3780         # Personalizar leyendas: borde negro y fondo blanco transparente
3781         axs[0].legend(loc='upper right', fontsize=7,bbox_to_anchor=(0.95,
0.95), edgecolor='black', facecolor='white', framealpha=0.5)
3782         axs[1].legend(loc='center', fontsize=7, bbox_to_anchor=(0.5, 0.8),
edgecolor='black', facecolor='white', framealpha=0.5)
3783
3784         # Actualizar los Labels con los valores dinámicos

```

```
3785     label_graficos_V.config(text=f"{cortante_y:.2f} kN")
3786     label_graficos_M.config(text=f"{-momento_y:.2f} kNm")
3787
3788
3789     canvas.draw()
3790
3791     slider_valor.config(command=actualizar_marcadores)
3792
3793
3794     # Asociar evento después de crear los widgets
3795     combo_graficos_Wu.bind("<<ComboboxSelected>>", mostrar_grafico)
3796
3797 def copiar_imagen_graficos():
3798     global fig # Asegúrate de que fig esté definido en el contexto
3799
3800     # Paso 1: Guardar el gráfico como una imagen en la carpeta especificada
3801     image_path = "Reportes/imagenes_pilot/grafico.png"
3802     fig.savefig(image_path, format='png', dpi=300, transparent=True)
3803
3804
3805 #Crear widgets para hacer saltos de codigo
3806 # combobox para momentos ultimos de viga
3807 Momento_Mu= ttk.Combobox(ventana)
3808 #Momento_Mu.place(x=500,y=600)
3809
3810 # combobox para barras de refuerzo de viga
3811 Barras_refuerzo= ttk.Combobox(ventana)
3812 #Barras_refuerzo.place(x=500,y=630)
3813
3814 # combobox para el #barras tracción
3815 Barras_traccion= ttk.Combobox(ventana)
3816 #Barras_traccion.place(x=500,y=660)
3817
3818 # combobox para el #barras compresión
3819 Barras_compresion= ttk.Combobox(ventana)
3820 #Barras_compresion.place(x=500,y=690)
3821
3822
3823 # Función para actualizar el Combobox con los valores de los labels
3824 def actualizar_combobox(tipo_viga):
3825     if tipo_viga == "Viga apoyada":
3826         # Solo actualizar el combobox con el valor de Mmax en caso de viga apoyada
3827         valores_Mu = [label_valor_Mmax.cget("text")]
3828     elif tipo_viga == "Viga empotrada":
3829         # Obtener los valores de M1, M2
3830         valor_M1 = label_valor_M1.cget("text")
3831         valor_M2 = label_valor_M2.cget("text")
3832
3833         # Obtener el valor de M3, eliminar solo el signo negativo y mantener el
3834         texto "kNm"
3835         valor_M3 = label_valor_M3.cget("text")
3836         if valor_M3.startswith('-'):
3837             valor_M3 = valor_M3[1:] # Eliminar el primer carácter (el signo
3838             negativo)
3839
3840         # Actualizar los valores para viga empotrada
3841         valores_Mu = [valor_M1, valor_M2, valor_M3]
3842
3843     # Actualizar las opciones del combobox
3844     Momento_Mu['values'] = valores_Mu
```

```
3843     if valores_Mu:
3844         Momento_Mu.set(valores_Mu[0]) # Establecer el valor por defecto en el
primer elemento
3845
3846
3847 def exportar_datos_a_excel():
3848     global datos_exportar
3849     global fig
3850     if not datos_exportar:
3851         print("No hay datos para exportar.")
3852         return
3853
3854     # Convertir los datos en un DataFrame
3855     df = pd.DataFrame(datos_exportar)
3856
3857     # Definir el directorio y archivo de salida
3858     directorio = "Archivos_Excel"
3859     if not os.path.exists(directorio):
3860         os.makedirs(directorio)
3861     archivo_salida = os.path.join(directorio, "datos_graficos.xlsx")
3862
3863     # Guardar el DataFrame en un archivo Excel
3864     df.to_excel(archivo_salida, index=False)
3865     print(f"Datos exportados correctamente a {archivo_salida}")
3866
3867     # Abrir el archivo Excel en el programa predeterminado
3868     if os.name == 'nt': # Si es Windows
3869         os.startfile(archivo_salida)
3870     elif os.name == 'posix': # Si es MacOS o Linux
3871         os.system(f'open "{archivo_salida}"') # Para MacOS
3872         # os.system(f'xdg-open "{archivo_salida}"') # Para Linux
3873
3874 # Definir las variables globales necesarias
3875 cortante = [] # Lista de valores de cortante
3876 posiciones_cortante = [] # Lista de posiciones para el cortante
3877 momento = [] # Lista de valores de momento
3878 x_momento = [] # Lista de posiciones para el momento
3879 Luz = None # Longitud de la viga
3880 label_frame_graficos_V_M = None # Definir esto según sea necesario
3881 slider_valor = None # Definir esto según sea necesario
3882 label_graficos_V = None # Definir esto según sea necesario
3883 label_graficos_M = None # Definir esto según sea necesario
3884 # Variable global para la figura
3885 fig = None
3886
3887
3888
3889 # Función para actualizar los valores de Momento_Mu
3890 def actualizar_momento_mu():
3891     # Cambiar los valores de Momento_Mu (esto debe venir de algún evento)
3892     nuevos_valores = Momento_Mu['values'] # Tomamos los mismos valores de
Momento_Mu
3893
3894     # Si la ventana flexión está abierta, actualizamos el ComboBox de esa ventana
3895     if ventana_flexion and ventana_flexion.wininfo_exists():
3896         # Aquí sincronizamos la actualización de los valores de combo_flexion_Mu
3897         combo_flexion_Mu['values'] = nuevos_valores # Actualizamos los valores de
combo_flexion_Mu
3898         if nuevos_valores:
```

```

3899         combo_flexion_Mu.set(nuevos_valores[0]) # Establecemos el primer
valor como valor por defecto
3900
3901
3902 def cerrar_ventana_flexion():
3903     ventana_flexion.destroy() # Cierra la ventana de flexión
3904
3905 # Variable global para almacenar la referencia de la ventana flexión
3906 ventana_flexion = None
3907
3908 # Variables globales para los Entry
3909 entry_flexion_VG = None
3910 entry_flexion_base = None
3911 entry_flexion_altura = None
3912 entry_flexion_recubrimiento = None
3913 entry_flexion_profundidad = None
3914
3915 # Función para abrir la ventana flexión
3916 def abrir_ventana_flexion():
3917     global ventana_flexion, entry_flexion_VG, entry_flexion_base,
entry_flexion_altura,entry_flexion_recubrimiento, entry_flexion_profundidad
3918     global combo_flexion_fc, combo_flexion_fy,
entry_flexion_β1,entry_flexion_cuantia_min, entry_flexion_cuantia_max
3919     global combo_flexion_Mu,combo_flexion_Rn, combo_flexion_preq,
combo_flexion_Asreq
3920     global datos_agregados
3921     datos_agregados = False
3922
3923     # Si la ventana flexión ya está abierta, no abrir otra
3924     if ventana_flexion is None or not ventana_flexion.winfo_exists():
3925         # Crear una nueva ventana secundaria usando Toplevel
3926         ventana_flexion = tk.Toplevel()
3927
3928         # Configurar la ventana
3929         ventana_flexion.title("Diseño a flexión")
3930         ventana_flexion.geometry("850x580")
3931
3932         # Hacer que la ventana no sea redimensionable
3933         ventana_flexion.resizable(False, False)
3934
3935         # Mantener la ventana siempre en primer plano
3936         ventana_flexion.attributes('-topmost', True)
3937
3938         #Crear contenido ventana flexion
3939         labelframe_flexion = tk.LabelFrame(ventana_flexion,text="")
3940         labelframe_flexion.place(x=20,y=20,width=810,height=510)
3941         labelframe_flexion_Mu = tk.LabelFrame(labelframe_flexion,text="Momento
ultimo/Sección transversal")
3942         labelframe_flexion_Mu.place(x=20,y=10,width=350,height=100)
3943         label_flexion_Mu = tk.Label(labelframe_flexion_Mu,text="Momento actuante
[Mu]:")
3944         label_flexion_Mu.place(x=20,y=15)
3945         combo_flexion_Mu = ttk.Combobox(labelframe_flexion_Mu)
3946         combo_flexion_Mu.place(x=180,y=15)
3947         combo_flexion_Mu['values'] = Momento_Mu['values']
3948         combo_flexion_Mu.set(Momento_Mu.get())
3949         label_flexion_VG = tk.Label(labelframe_flexion_Mu,text="Sección elemento
[VG]:")
3950         label_flexion_VG.place(x=20,y=45)
3951         entry_flexion_VG = tk.Entry(labelframe_flexion_Mu)

```

```

3952     entry_flexion_VG.place(x=180,y=45,width=143)
3953     entry_flexion_VG.insert(0, label_seccion_cargas.cget("text"))
3954     entry_flexion_VG.bind("<KeyRelease>", actualizar_dimensiones)
3955
3956
3957     # Crear StringVars para los Entry
3958     base_var = tk.StringVar()
3959     recubrimiento_var = tk.StringVar()
3960
3961     labelframe_flexion_dimensiones = tk.LabelFrame(labelframe_flexion
, text="Dimensiones de la viga [VG]")
3962     labelframe_flexion_dimensiones.place(x=20,y=130,width=350,height=110)
3963     label_flexion_base=tk.Label(labelframe_flexion_dimensiones, text="Base
[b]:")
3964     label_flexion_base.place(x=20,y=15)
3965     entry_flexion_base= tk.Entry(labelframe_flexion_dimensiones,
textvariable=base_var)
3966     entry_flexion_base.place(x=110,y=15,width=143)
3967     label_flexion_unidad_b= tk.Label(labelframe_flexion_dimensiones, text="
[m]")
3968     label_flexion_unidad_b.place(x=280,y=15)
3969     label_flexion_altura =
tk.Label(labelframe_flexion_dimensiones, text="Altura [h]:")
3970     label_flexion_altura.place(x=20,y=45)
3971     entry_flexion_altura= tk.Entry(labelframe_flexion_dimensiones)
3972     entry_flexion_altura.place(x=110,y=45,width=143)
3973     label_flexion_unidad_h= tk.Label(labelframe_flexion_dimensiones, text="
[m]")
3974     label_flexion_unidad_h.place(x=280,y=45)
3975     labelframe_flexion_r_d =
tk.LabelFrame(labelframe_flexion, text="Recubrimiento/Altura efectiva")
3976     labelframe_flexion_r_d.place(x=20,y=260,width=350,height=110)
3977     label_flexion_recubrimiento =
tk.Label(labelframe_flexion_r_d, text="Recubrimiento [r]:")
3978     label_flexion_recubrimiento.place(x=5,y=15)
3979     entry_flexion_recubrimiento=
tk.Entry(labelframe_flexion_r_d, textvariable=recubrimiento_var)
3980     entry_flexion_recubrimiento.place(x=110,y=15,width=143)
3981     entry_flexion_recubrimiento.bind("<KeyRelease>", actualizar_profundidad)
3982     label_flexion_unidad_r= tk.Label(labelframe_flexion_r_d, text="[m]")
3983     label_flexion_unidad_r.place(x=280,y=15)
3984     label_flexion_profundidad =
tk.Label(labelframe_flexion_r_d, text="Profundidad [d]:")
3985     label_flexion_profundidad.place(x=5,y=45)
3986     entry_flexion_profundidad= tk.Entry(labelframe_flexion_r_d)
3987     entry_flexion_profundidad.place(x=110,y=45,width=143)
3988     label_flexion_unidad_d= tk.Label(labelframe_flexion_r_d, text="[m]")
3989     label_flexion_unidad_d.place(x=280,y=45)
3990     labelframe_flexion_info=
tk.LabelFrame(labelframe_flexion, text="Información adicional")
3991     labelframe_flexion_info.place(x=20,y=380,width=350,height=100)
3992     label_flexion_NSR10 = tk.Label(labelframe_flexion_info, text="Norma de
diseño:")
3993     label_flexion_NSR10.place(x=5,y=15)
3994     L_flexion_NSR10 = tk.Label(labelframe_flexion_info, text="NSR-10")
3995     L_flexion_NSR10.place(x=110, y=15, width=143)
3996     label_flexion_estribo= tk.Label(labelframe_flexion_info, text="Estribo de
diseño:")
3997     label_flexion_estribo.place(x=5,y=45)
3998

```

```

3999     labelframe_flexion_materiales=
tk.LabelFrame(labelframe_flexion,text="Resistencia de materiales [ $\sigma$ ]")
4000     labelframe_flexion_materiales.place(x=380,y=10,width=408,height=100)
4001     label_flexion_fc= tk.Label(labelframe_flexion_materiales,text="Concreto
[f'c]:")
4002     label_flexion_fc.place(x=5,y=15)
4003     combo_flexion_fc= ttk.Combobox(labelframe_flexion_materiales)
4004     combo_flexion_fc.place(x=95,y=15)
4005     combo_flexion_fc['values'] = ["Seleccionar", "17.5 MPa", "21 MPa", "24.5
MPa", "28 MPa", "35 MPa", "42 MPa"]
4006     combo_flexion_fc.current(0)
4007     combo_flexion_fc.bind("<<ComboboxSelected>>", lambda event:
actualizar_β1(event, combo_flexion_fc, combo_flexion_fy, entry_flexion_β1,
4008     entry_flexion_cuantia_min,
entry_flexion_cuantia_max,combo_flexion_Rn,combo_flexion_preq,
4009
    combo_flexion_Asreq,entry_flexion_base,entry_flexion_profundidad,Barras_refuerzo,
label_fc,
4010
    label_cuantia_minima,label_cuantia_maxima,label_beta1,label_cuantia_requerida,lab
el_acero_requerido,
4011
    label_As2,label_req_C2))
4012
4013
4014
4015
4016     label_flexion_fy= tk.Label(labelframe_flexion_materiales,text="Acero
[fy]:")
4017     label_flexion_fy.place(x=5,y=45)
4018     combo_flexion_fy= ttk.Combobox(labelframe_flexion_materiales)
4019     combo_flexion_fy.place(x=95,y=45)
4020     combo_flexion_fy['values'] = ["Seleccionar", "420 MPa"]
4021     combo_flexion_fy.current(1)
4022     label_flexion_β1 = tk.Label(labelframe_flexion_materiales,text="Factor
[β1]:")
4023     label_flexion_β1.place(x=280,y=15)
4024     entry_flexion_β1=tk.Entry(labelframe_flexion_materiales)
4025     entry_flexion_β1.place(x=273,y=45,width=80)
4026
4027     labelframe_flexion_cuantia =
tk.LabelFrame(labelframe_flexion,text="Cuantia diseño [ $\rho$ ]")
4028     labelframe_flexion_cuantia.place(x=380,y=130,width=180,height=110)
4029     label_flexion_cuantia_min =
tk.Label(labelframe_flexion_cuantia,text="pmin:")
4030     label_flexion_cuantia_min.place(x=10,y=10)
4031     entry_flexion_cuantia_min = tk.Entry(labelframe_flexion_cuantia)
4032     entry_flexion_cuantia_min.place(x=70,y=10,width=90)
4033     label_flexion_cuantia_max =
tk.Label(labelframe_flexion_cuantia,text="pmax:")
4034     label_flexion_cuantia_max.place(x=10,y=50)
4035     entry_flexion_cuantia_max = tk.Entry(labelframe_flexion_cuantia)
4036     entry_flexion_cuantia_max.place(x=70,y=50,width=90)
4037
4038     labelframe_flexion_calculos =
tk.LabelFrame(labelframe_flexion,text="Cálculos previos")
4039     labelframe_flexion_calculos.place(x=570,y=130,width=218,height=110)
4040     label_flexion_Rn = tk.Label(labelframe_flexion_calculos,text="Rn:")
4041     label_flexion_Rn.place(x=30,y=10)
4042     combo_flexion_Rn = ttk.Combobox(labelframe_flexion_calculos)
4043     combo_flexion_Rn.place(x=80,y=10,width=90)

```

```

4044 label_flexion_preq = tk.Label(labelframe_flexion_calculos,text="preq:")
4045 label_flexion_preq.place(x=30,y=35)
4046 combo_flexion_preq = ttk.Combobox(labelframe_flexion_calculos)
4047 combo_flexion_preq.place(x=80,y=35,width=90)
4048 label_flexion_Asreq = tk.Label(labelframe_flexion_calculos,text="Asreq:")
4049 label_flexion_Asreq.place(x=30,y=60)
4050 combo_flexion_Asreq = ttk.Combobox(labelframe_flexion_calculos)
4051 combo_flexion_Asreq.place(x=80,y=60,width=90)
4052
4053 # Cuando cambie la base o el recubrimiento, actualizamos los entrys
externos
4054 def actualizar_entrys_externos(*args):
4055     # Actualizar el entry externo para base
4056     entry_prueba_base.delete(0, tk.END)
4057     entry_prueba_base.insert(0, base_var.get())
4058
4059     # Actualizar el entry externo para recubrimiento, multiplicar por 1000
y añadir "mm"
4060     try:
4061         recubrimiento_value = float(recubrimiento_var.get()) * 1000 #
Convertir a mm
4062         entry_prueba_recubrimiento.delete(0, tk.END)
4063         entry_prueba_recubrimiento.insert(0, f"{recubrimiento_value} mm")
4064     except ValueError:
4065         entry_prueba_recubrimiento.delete(0, tk.END)
4066         entry_prueba_recubrimiento.insert(0, "Error")
4067
4068     # Vincular el trace a las variables para actualizar automáticamente
4069     base_var.trace("w", actualizar_entrys_externos)
4070     recubrimiento_var.trace("w", actualizar_entrys_externos)
4071
4072
4073 boton_aceptar_flexion =
ttk.Button(ventana_flexion,text="Aceptar",command=cerrar_ventana_flexion)
4074 boton_aceptar_flexion.place(x=380,y=540,width=100)
4075
4076 # Configurar un estilo personalizado para el Notebook
4077 style = ttk.Style()
4078 style.configure("CustomNotebook.TNotebook", background="lightgray",
tabmargins=[0, 0, 0, 0])
4079 style.configure("CustomNotebook.TNotebook.Tab", background="white",
foreground="black", padding=[10, 2])
4080 style.map("CustomNotebook.TNotebook.Tab", background=[("selected",
"lightgray")])
4081
4082 frame_flexion_tabla = tk.Frame(labelframe_flexion,bd=2,relief="groove")
4083 frame_flexion_tabla.place(x=380,y=268,width=409,height=212)
4084
4085 # Crear el notebook_tabla con el estilo personalizado
4086 notebook_tabla = ttk.Notebook(frame_flexion_tabla,
style="CustomNotebook.TNotebook")
4087
4088 # Crear la pestaña "Tabla"
4089 tab1 = ttk.Frame(notebook_tabla)
4090 notebook_tabla.add(tab1, text="Resultados")
4091
4092 # Crear el Treeview (tabla de datos) dentro de la pestaña
4093 columns = ("Mu [kNm]", "Rn [kN/m²]", "Cuantia\nDiseño", "As")
4094 treeview = ttk.Treeview(tab1, columns=columns, show="headings")
4095

```

```

4096
4097 # Configurar las columnas del Treeview
4098 treeview.heading("Mu [kNm]", text="Mu [kNm]")
4099 treeview.heading("Rn [kN/m²]", text="Rn [kN/m²]")
4100 treeview.heading("Cuantia\nDiseño", text="Cuantia")
4101 treeview.heading("As", text="As [mm²]")
4102
4103 # Configurar el ancho de cada columna individualmente
4104 treeview.column("Mu [kNm]", width=80,anchor="center")
4105 treeview.column("Rn [kN/m²]", width=80,anchor="center")
4106 treeview.column("Cuantia\nDiseño", width=80,anchor="center")
4107 treeview.column("As", width=80,anchor="center")
4108
4109
4110 treeview.place(x=10, y=35, width=380, height=90)
4111
4112
4113 # Variables para almacenar los valores seleccionados
4114 valores_mu = []
4115 valores_rn = []
4116 valores_cuantia = []
4117 valores_as = []
4118
4119 # Función para insertar datos en el Treeview al presionar el botón
4120 def insertar_todos_los_datos():
4121     global valores_mu, valores_rn, valores_cuantia, valores_as
4122     # Obtener valores seleccionados de los combobox
4123     valores_mu = [valor.split()[0] for valor in
combo_flexion_Mu['values']]
4124     valores_rn = combo_flexion_Rn['values']
4125     valores_cuantia = combo_flexion_preq['values']
4126     valores_as = combo_flexion_Asreq['values']
4127
4128     # Limpiar las filas del Treeview antes de agregar nuevos datos
4129     for item in treeview.get_children():
4130         treeview.delete(item)
4131
4132     # Crear las filas combinando los valores
4133     for i in range(len(valores_mu)):
4134         treeview.insert("", "end", values=(valores_mu[i], valores_rn[i],
valores_cuantia[i], valores_as[i]))
4135
4136
4137     boton_insertar= ttk.Button(tab1,text="Agregar
datos",command=insertar_todos_los_datos)
4138     boton_insertar.place(x=80,y=140)
4139
4140
4141 # Función para verificar los datos
4142 def verificar_datos():
4143     # Obtener los valores de valores_mu y valores_cuantia (estos son los
valores de los combobox)
4144     valores_mu = [valor.split()[0] for valor in
combo_flexion_Mu['values']]
4145     valores_cuantia = combo_flexion_preq['values']
4146
4147     # Verificar que se han ingresado 1 o 3 valores de gamma y valores_mu
4148     if len(valores_cuantia) not in [1, 3] or len(valores_mu) not in [1,
3]:

```

```

4149         print("Error: Se deben ingresar 1 o 3 valores tanto para Mu como
para gamma.")
4150         return
4151
4152         # Obtener los valores necesarios para el cálculo de  $\phi Mn$ 
4153         phi = 0.90 # Constante para  $\phi$ 
4154         # Extraer solo el valor numérico de fy
4155         fy_flexion = combo_flexion_fy.get().split()[0]
4156         fy_flexion = float(fy_flexion) # Convertir a float
4157
4158         # Extraer solo el valor numérico de f'c
4159         fc_flexion = combo_flexion_fc.get().split()[0]
4160         fc_flexion = float(fc_flexion) # Convertir a float
4161
4162         gamma_values = [float(value) for value in valores_cuantia]
4163         beta = float(entry_flexion_beta.get()) # Valor de  $\beta$ 
4164         b_flexion = float(entry_flexion_base.get()) # Valor de b
4165         d_flexion = float(entry_flexion_profundidad.get()) # Valor de d
4166
4167
4168         # Calcular  $\phi Mn$  para cada valor de gamma
4169         resultados_phiMn = []
4170         for gamma in gamma_values:
4171             try:
4172                 # Fórmula corregida para  $\phi Mn$ :
4173                 numerador = gamma * fy_flexion
4174                 fraccion = (numerador) / (2 * beta * fc_flexion) # La
fracción dentro del corchete
4175                 phiMn = phi * numerador * 1000 * (1 - fraccion) * b_flexion *
d_flexion**2
4176                 resultados_phiMn.append(phiMn)
4177             except ZeroDivisionError:
4178                 resultados_phiMn.append(0)
4179
4180         # Si solo hay 1 valor de gamma, mostrarlo en la primera fila y "N/A"
en las otras filas
4181         if len(gamma_values) == 1:
4182             resultados_phiMn = [resultados_phiMn[0], "N/A", "N/A"]
4183             valores_mu = [valores_mu[0], "N/A", "N/A"]
4184             valores_cuantia = [valores_cuantia[0], "N/A", "N/A"]
4185
4186         elif len(gamma_values) == 3:
4187             resultados_phiMn = resultados_phiMn[:3] # Solo tomar los tres
primeros resultados si hay tres valores de gamma
4188             valores_mu = valores_mu[:3]
4189             valores_cuantia = valores_cuantia[:3]
4190
4191         # Verificar que el número de filas en el treeview coincida con la
cantidad de datos
4192         cumple_diseño = None
4193         cumple_resultados = [] # Para almacenar si cumple o no cumple en cada
fila
4194
4195         # Verificar que el número de filas en el treeview coincida con la
cantidad de datos
4196         treeview_rows = len(treeview_flexion.get_children())
4197
4198         # Asegurarse de que el número de filas coincida con los valores a
insertar
4199         for i, item_id in enumerate(treeview_flexion.get_children()):

```

```

4200         # Si el índice i es menor que la longitud de los valores_mu, se
actualiza la fila
4201         if i < len(valores_mu):
4202             current_values = treeview_flexion.item(item_id)['values']
4203
4204         # Verificar si valores_mu[i] no es "N/A" antes de intentar convertirlo
a float
4205         if valores_mu[i] != "N/A" and float(valores_mu[i]) <=
resultados_φMn[i]:
4206             cumple = "Cumple"
4207         else:
4208             cumple = "No cumple"
4209
4210         # Actualizamos las columnas correspondientes
4211         new_values = (current_values[0], # "Mmax_Mmin"
4212                       valores_mu[i],      # "Mu_kNm"
4213                       valores_cuantia[i],  # "Cuantia"
4214                       resultados_φMn[i],   # "φMn"
4215                       cumple               # "Mu ≤ φMn"
4216                       )
4217         treeview_flexion.item(item_id, values=new_values)
4218
4219         # Almacenamos los resultados de "cumple" o "no cumple" para cada fila
4220         cumple_resultados.append(cumple)
4221
4222         # Actualizar el mensaje en los labels:
4223         if len(resultados_φMn) > 0:
4224             # Actualizar los valores de φMn en los labels
4225             label_φMn1.config(text=f"{resultados_φMn[0]:.4f} kNm") #
Primer valor
4226             if len(resultados_φMn) > 1:
4227                 label_φMn2.config(text=f"{resultados_φMn[1]:.4f} kNm") #
Segundo valor
4228
4229         # Actualizar el label de conclusión ( Si Cumple o No Cumple)
4230         if len(cumple_resultados) > 0:
4231             if cumple_resultados[0] == "Cumple":
4232                 label_conclusion_φMn.config(text="Si Cumple")
4233             else:
4234                 label_conclusion_φMn.config(text="No Cumple")
4235
4236         # Actualizar label_conclusion_φMn2 con el segundo dato de
cumple_resultados
4237         if len(cumple_resultados) > 1:
4238             if cumple_resultados[1] == "Cumple":
4239                 label_conclusion_φMn2.config(text="Si Cumple")
4240             else:
4241                 label_conclusion_φMn2.config(text="No cumple")
4242
4243         # Determinar el mensaje a mostrar en label_check
4244         if cumple == "Cumple":
4245             cumple_diseño = "Cumple"
4246         else:
4247             cumple_diseño = "No cumple"
4248
4249         # Limpiar las filas sobrantes si hay menos de 3 valores de gamma
4250         elif i >= len(valores_mu):
4251             current_values = treeview_flexion.item(item_id)['values']
4252             new_values = (current_values[0], # "Mmax_Mmin"
4253                           "N/A",           # "Mu_kNm"

```

```

4254         "N/A",                # "Cuantia"
4255         "N/A",                # "φMn"
4256         "N/A",                # "Mu ≤ φMn"
4257     )
4258     treeview_flexion.item(item_id, values=new_values)
4259
4260     # Verificar los resultados para actualizar el mensaje del label_check
4261     if len(gamma_values) == 1:
4262         # Para 1 valor de gamma, solo verificar el primer resultado
4263         if cumple_resultados[0] == "Cumple":
4264             cumple_diseño = "Cumple"
4265         else:
4266             cumple_diseño = "No cumple"
4267
4268     elif len(gamma_values) == 3:
4269         # Para 3 valores de gamma, verificar si todos cumplen
4270         if all(cumple == "Cumple" for cumple in cumple_resultados):
4271             cumple_diseño = "Cumple"
4272         else:
4273             cumple_diseño = "No cumple"
4274     # Actualizar el mensaje en label_check basado en el resultado
4275     if cumple_diseño == "Cumple":
4276         # Obtener los valores de combo_flexion_fc y
entry_flexion_recubrimiento
4277         fc_value = combo_flexion_fc.get() # Valor de f'c
4278         recubrimiento_value = entry_flexion_recubrimiento.get() # Valor
de recubrimiento
4279         label_check.config(
4280             text= f"El diseño cumple con los criterios de resistencia.
"
4281             f"Está bien dimensionado para soportar los momentos
flexionantes "
4282             f"sin superar la capacidad estructural, considerando un
f'c de {fc_value} "
4283             f"y un adecuado recubrimiento de {recubrimiento_value}m
que garantiza la seguridad de la estructura.",
4284             bg="lightgreen") # Cambia el color de fondo a verde
4285
4286     elif cumple_diseño == "No cumple":
4287         label_check.config(
4288             text="El diseño NO cumple. Podrían ocurrir problemas como
deformaciones, "
4289             "grietas o fallas por flexión. Se recomienda reconsiderar los
materiales "
4290             "(f'c) o rediseñar la sección transversal para aumentar la
capacidad de momento.",
4291             bg="lightcoral") # Cambia el color de fondo a rojo
4292
4293
4294     boton_verificar= ttk.Button(tab1,text="Verificar
datos",command=verificar_datos)
4295     boton_verificar.place(x=210,y=140)
4296
4297     # Función para evitar el redimensionamiento de las columnas
4298     def no_resize(event):
4299         # Restaurar el tamaño original de las columnas
4300         for col in treeview["columns"]:
4301             treeview.column(col, width=94)
4302

```

```
4303         # Vincular el evento para evitar la acción de cambiar el tamaño de las
columnas
4304         treeview.bind("<ButtonRelease-1>", no_resize)
4305
4306         # Mostrar el notebook_tabla dentro del frame
4307         notebook_tabla.pack(expand=True, fill="both")
4308
4309         entry_flexion_recubrimiento.bind("<FocusOut>", lambda event:
calcular_Rn_al_modificar_recubrimiento(
4310             event, combo_flexion_Mu, entry_flexion_base, entry_flexion_profundidad,
entry_flexion_recubrimiento, combo_flexion_Rn,
4311             label_Resistencia_Rn, label_Rn2))
4312
4313
4314         # Crear radiobutton No.3 No.4 con comandos para actualizar el valor
4315         radiobutton_flexion_No3 = tk.Radiobutton(labelframe_flexion_info,
text="No.3",
4316                                                     variable=estado_estribo,
4317                                                     value="9.5",
4318                                                     command=lambda:
actualizar_label_estribo(9.5))
4319         radiobutton_flexion_No3.place(x=150, y=45)
4320
4321         radiobutton_flexion_No4 = tk.Radiobutton(labelframe_flexion_info,
text="No.4",
4322                                                     variable=estado_estribo,
4323                                                     value="12.7",
4324                                                     command=lambda:
actualizar_label_estribo(12.7))
4325         radiobutton_flexion_No4.place(x=250, y=45)
4326
4327         entry_prueba_base= tk.Entry(ventana)
4328         #entry_prueba_base.place(x=500,y=700)
4329
4330         entry_prueba_recubrimiento = tk.Entry(ventana)
4331         #entry_prueba_recubrimiento.place(x=500,y=730)
4332
4333         def calcular_Rn_al_modificar_recubrimiento(event=None, combo_flexion_Mu=None,
entry_flexion_base=None, entry_flexion_profundidad=None,
entry_flexion_recubrimiento=None, combo_flexion_Rn=None,
4334         label_Resistencia_Rn=None, label_Rn2=None):
4335             try:
4336                 # Leer el valor de b desde el Entry (base)
4337                 b = float(entry_flexion_base.get())
4338
4339                 # Leer el valor de d desde el Entry (profundidad)
4340                 d = float(entry_flexion_profundidad.get())
4341
4342                 # Leer el valor del recubrimiento
4343                 recubrimiento = float(entry_flexion_recubrimiento.get())
4344
4345                 # Validar que d sea positivo
4346                 if d <= 0:
4347                     raise ValueError("La profundidad (d) debe ser positiva.")
4348
4349                 # Leer los valores de Mu desde el combobox (como lista de texto)
4350                 mu_values = combo_flexion_Mu['values']
4351                 mu_numeric_values = []
4352
```

```

4353     for mu in mu_values:
4354         try:
4355             # Extraer el valor numérico, ignorando "kNm"
4356             mu_numeric_values.append(float(mu.split()[0]))
4357         except ValueError:
4358             pass # Ignorar valores no válidos
4359
4360     # Constante de diseño
4361     phi = 0.90
4362
4363     # Calcular Rn para cada Mu
4364     rn_values = []
4365     for mu in mu_numeric_values:
4366         try:
4367             rn = mu / (phi * b * (d ** 2))
4368             rn_values.append(f"{rn:.4f}")
4369         except ZeroDivisionError:
4370             rn_values.append("Error")
4371
4372     # Actualizar el combo_flexion_Rn con los resultados
4373     combo_flexion_Rn['values'] = rn_values
4374
4375     # Si hay resultados, seleccionar el primero por defecto
4376     if rn_values:
4377         combo_flexion_Rn.set(rn_values[0])
4378     else:
4379         combo_flexion_Rn.set("")
4380
4381     # Actualizar el label_Resistencia_Rn con el valor calculado (primer valor
de la lista)
4382     if rn_values:
4383         label_Resistencia_Rn.config(text=f"{rn_values[0]}")
4384     else:
4385         label_Resistencia_Rn.config(text="Rn: Error")
4386
4387     # Actualizar el label_Rn2 con el segundo valor de la lista (si existe)
4388     if len(rn_values) > 1:
4389         label_Rn2.config(text=f"{rn_values[1]}")
4390     else:
4391         label_Rn2.config(text="Rn2: No disponible")
4392
4393     except ValueError as e:
4394         # Manejar entradas no válidas
4395         combo_flexion_Rn['values'] = []
4396         combo_flexion_Rn.set("")
4397         label_Resistencia_Rn.config(text="Rn: Error")
4398         label_Rn2.config(text="Rn2: Error")
4399         print(f"Error: {e}")
4400
4401 def actualizar_β1(
4402     event,
4403     combo_flexion_fc,
4404     combo_flexion_fy,
4405     entry_flexion_β1,
4406     entry_flexion_cuantia_min,
4407     entry_flexion_cuantia_max,
4408     combo_flexion_Rn,
4409     combo_flexion_preq,
4410     combo_flexion_Asreq,
4411     entry_flexion_base,

```

```

4412     entry_flexion_profundidad,
4413     Barras_refuerzo,
4414     label_fc,
4415     label_cuantia_minima,
4416     label_cuantia_maxima,
4417     label_beta1,
4418     label_cuantia_requerida,
4419     label_acero_requerido,
4420     label_As2,
4421     label_req_C2
4422 ):
4423
4424     # Obtener los valores de f'c y fy desde los combos
4425     fc_value = combo_flexion_fc.get()
4426     fy_value = combo_flexion_fy.get()
4427
4428     # Actualizar el label_fc con el valor de f'c seleccionado
4429     label_fc.config(text=f"{fc_value}")
4430
4431     # Convertir los valores a solo los números (ignorando "MPa")
4432     try:
4433         fc_numeric = float(fc_value.split()[0]) # Extraer el número antes de
"MPa"
4434     except ValueError:
4435         fc_numeric = 0.0
4436
4437     try:
4438         fy_numeric = float(fy_value.split()[0]) # Extraer el número antes de
"MPa"
4439     except ValueError:
4440         fy_numeric = 0.0
4441
4442     # Asignar el valor de  $\beta_1$  basado en f'c
4443     if fc_numeric == 17.5 or fc_numeric == 21 or fc_numeric == 24.5 or fc_numeric
== 28:
4444          $\beta_1$  = 0.85
4445     elif fc_numeric == 35:
4446          $\beta_1$  = 0.80
4447     elif fc_numeric == 42:
4448          $\beta_1$  = 0.75
4449     else:
4450          $\beta_1$  = None
4451
4452     if  $\beta_1$  is not None:
4453         entry_flexion_ $\beta_1$ .delete(0, tk.END)
4454         entry_flexion_ $\beta_1$ .insert(0, f"{ $\beta_1$ :.2f}")
4455         # Actualizar el valor de  $\beta_1$  en el label_beta1
4456         label_beta1.config(text=f"{ $\beta_1$ :.2f}")
4457     else:
4458         entry_flexion_ $\beta_1$ .delete(0, tk.END)
4459         label_beta1.config(text="No Definido")
4460
4461     # Calcular cuantía mínima:  $p_{min} = (0.25 * \sqrt{f'c}) / f_y$ 
4462     if fc_numeric > 0 and fy_numeric > 0:
4463         cuantia_min = max((0.25 * math.sqrt(fc_numeric)) / fy_numeric, 0.0033)
4464         entry_flexion_cuantia_min.delete(0, tk.END)
4465         entry_flexion_cuantia_min.insert(0, f"{cuantia_min:.4f}")
4466         # Actualizar el valor de cuantía mínima en el label_cuantia_minima
4467         label_cuantia_minima.config(text=f"{cuantia_min:.4f}")
4468     else:

```

```

4469     entry_flexion_cuantia_min.delete(0, tk.END)
4470     label_cuantia_minima.config(text="No Definida")
4471
4472     # Calcular cuantía máxima:  $p_{max} = 0.85 * \beta_1 * (f'_c / f_y) * 0.63$ 
4473     if  $\beta_1$  is not None and fc_numeric > 0 and fy_numeric > 0:
4474         cuantia_max =  $0.85 * \beta_1 * (fc\_numeric / fy\_numeric) * 0.63$ 
4475         entry_flexion_cuantia_max.delete(0, tk.END)
4476         entry_flexion_cuantia_max.insert(0, f"{cuantia_max:.4f}")
4477         # Actualizar el valor de cuantía máxima en el label_cuantia_maxima
4478         label_cuantia_maxima.config(text=f"{cuantia_max:.4f}")
4479     else:
4480         entry_flexion_cuantia_max.delete(0, tk.END)
4481         label_cuantia_maxima.config(text="No Definida")
4482
4483     # Obtener los valores de Rn desde el combo
4484     rn_values = combo_flexion_Rn['values']
4485
4486     # Calcular cuantía requerida:  $prequerida = 0.85 * f'_c / f_y * (1 - \sqrt{1 - 2 * R_n * 0.001 / (0.85 * f'_c)})$ 
4487     cuantia_req_values = []
4488     for rn_value in rn_values:
4489         try:
4490             rn_numeric = float(rn_value)
4491             if fc_numeric > 0 and fy_numeric > 0:
4492                 cuantia_req =  $0.85 * fc\_numeric / fy\_numeric * (1 - \sqrt{1 - 2 * rn\_numeric * 0.001 / (0.85 * fc\_numeric)})$ 
4493             )
4494             cuantia_req_values.append(f"{cuantia_req:.4f}")
4495         except (ValueError, ZeroDivisionError):
4496             cuantia_req_values.append("Error")
4497
4498     # Actualizar el combo_flexion_preq con los resultados
4499     combo_flexion_preq['values'] = cuantia_req_values
4500
4501     # Seleccionar el primer valor si hay resultados
4502     if cuantia_req_values:
4503         combo_flexion_preq.set(cuantia_req_values[0])
4504     else:
4505         combo_flexion_preq.set("")
4506
4507     # Cálculo del acero requerido  $A_s = p_{req} * base\_As * d\_As$ 
4508     try:
4509         base_As = float(entry_flexion_base.get())
4510         d_As = float(entry_flexion_profundidad.get())
4511     except ValueError:
4512         base_As, d_As = 0.0, 0.0
4513
4514     acero_req_values = []
4515     for cuantia_req_str in cuantia_req_values:
4516         try:
4517             cuantia_req = float(cuantia_req_str)
4518             As = cuantia_req * (base_As * 1000) * (d_As * 1000)
4519             acero_req_values.append(f"{As:.3f}")
4520         except ValueError:
4521             acero_req_values.append("Error")
4522
4523     # Actualizar el combo_flexion_Asreq con los resultados
4524     combo_flexion_Asreq['values'] = acero_req_values
4525
4526

```

```
4527 # Seleccionar el primer valor si hay resultados
4528 if acero_req_values:
4529     combo_flexion_Asreq.set(acero_req_values[0])
4530 else:
4531     combo_flexion_Asreq.set("")
4532
4533 # ACTUALIZAR 'Barras_refuerzo' CON LOS RESULTADOS DE 'combo_flexion_Asreq'
4534 Barras_refuerzo['values'] = acero_req_values
4535 if acero_req_values:
4536     Barras_refuerzo.set(acero_req_values[0]) # Seleccionar el primer valor si
hay resultados
4537 else:
4538     Barras_refuerzo.set("")
4539
4540 # Actualizar el valor de acero requerido en el label_acero_requerido
(completar con el valor de combo_flexion_Asreq)
4541 if acero_req_values:
4542     label_acero_requerido.config(text=f"{acero_req_values[0]}")
4543 else:
4544     label_acero_requerido.config(text="No Definido")
4545
4546 # Actualizar el valor de cuantía requerida en el label_cuantia_requerida
(completar con el valor de combo_flexion_preq)
4547 if cuantia_req_values:
4548     label_cuantia_requerida.config(text=f"{cuantia_req_values[0]}")
4549 else:
4550     label_cuantia_requerida.config(text="No Definida")
4551
4552 # ACTUALIZAR label_As2 CON EL SEGUNDO VALOR DE 'acero_req_values'
4553 if len(acero_req_values) > 1:
4554     label_As2.config(text=f"{acero_req_values[1]}")
4555 else:
4556     label_As2.config(text="No Definido")
4557
4558 # ACTUALIZAR label_req_C2 CON EL SEGUNDO VALOR DE 'cuantia_req_values'
4559 if len(cuantia_req_values) > 1:
4560     label_req_C2.config(text=f"{cuantia_req_values[1]}")
4561 else:
4562     label_req_C2.config(text="No Definido")
4563
4564
4565 # Función para actualizar las dimensiones
4566 def actualizar_dimensiones(event):
4567     global entry_flexion_base, entry_flexion_altura, entry_flexion_VG,
entry_flexion_recubrimiento, entry_flexion_profundidad, label_h
4568     global label_recubrimiento, label_altura_efectiva
4569
4570
4571     vg_text = entry_flexion_VG.get()
4572
4573     if "X" in vg_text:
4574         try:
4575
4576             dimensiones_1 = vg_text.split("X")
4577             base_1 = float(dimensiones_1[0].strip().split()[1])
4578             altura_1 = float(dimensiones_1[1].strip())
4579
4580             # Multiplicar por 0.01 para convertir a metros
4581             base_resultado = base_1 * 0.01
4582             altura_resultado = altura_1 * 0.01
```

```

4583
4584
4585     base_resultado = round(base_resultado, 2)
4586     altura_resultado = round(altura_resultado, 2)
4587
4588
4589     entry_flexion_base.delete(0, tk.END)
4590     entry_flexion_base.insert(0, str(base_resultado))
4591
4592     entry_flexion_altura.delete(0, tk.END)
4593     entry_flexion_altura.insert(0, str(altura_resultado))
4594
4595
4596     label_h.config(text=f"{altura_resultado:.2f} m")
4597
4598     recubrimiento_1 = float(entry_flexion_recubrimiento.get()) if
entry_flexion_recubrimiento.get() else 0.0
4599
4600
4601     recubrimiento_cm = recubrimiento_1 * 100
4602     label_recubrimiento.config(text=f"{recubrimiento_cm:.2f} cm")
4603
4604
4605     altura_d = (altura_1 - recubrimiento_cm) * 0.01
4606
4607
4608     altura_d = round(altura_d, 2)
4609
4610
4611     entry_flexion_profundidad.delete(0, tk.END)
4612     entry_flexion_profundidad.insert(0, str(altura_d))
4613
4614
4615     label_altura_efectiva.config(text=f"{altura_d:.2f} m")
4616
4617     except ValueError:
4618
4619         pass
4620
4621
4622 # Función para cerrar la ventana y permitir abrirla nuevamente
4623 def cerrar_ventana_barras():
4624     global ventana_refuerzo_abierta
4625     ventana_refuerzo_abierta = False
4626     ventana_refuerzo.destroy()
4627
4628 # Diccionario con la información de las barras de refuerzo
4629 barras_info_refuerzo = {
4630     "Barra No.2": {"diametro_pulg": "1/4", "diametro_mm": 6.4, "area_mm2": 32,
"perimetro_mm": 20.0, "masa_kg_m": 0.250},
4631     "Barra No.3": {"diametro_pulg": "3/8", "diametro_mm": 9.5, "area_mm2": 71,
"perimetro_mm": 30.0, "masa_kg_m": 0.560},
4632     "Barra No.4": {"diametro_pulg": "1/2", "diametro_mm": 12.7, "area_mm2": 129,
"perimetro_mm": 40.0, "masa_kg_m": 0.994},
4633     "Barra No.5": {"diametro_pulg": "5/8", "diametro_mm": 15.9, "area_mm2": 199,
"perimetro_mm": 50.0, "masa_kg_m": 1.552},
4634     "Barra No.6": {"diametro_pulg": "3/4", "diametro_mm": 19.1, "area_mm2": 284,
"perimetro_mm": 60.0, "masa_kg_m": 2.235},
4635     "Barra No.7": {"diametro_pulg": "7/8", "diametro_mm": 22.2, "area_mm2": 387,
"perimetro_mm": 70.0, "masa_kg_m": 3.042},

```

```

4636     "Barra No.8": {"diametro_pulg": "1", "diametro_mm": 25.4, "area_mm2": 510,
"perimetro_mm": 80.0, "masa_kg_m": 3.973},
4637     "Barra No.9": {"diametro_pulg": "1-1/8", "diametro_mm": 28.7, "area_mm2": 645,
"perimetro_mm": 90.0, "masa_kg_m": 5.060},
4638     "Barra No.10": {"diametro_pulg": "1-1/4", "diametro_mm": 32.3, "area_mm2":
819, "perimetro_mm": 101.3, "masa_kg_m": 6.404},
4639     "Barra No.11": {"diametro_pulg": "1-3/8", "diametro_mm": 35.8, "area_mm2":
1006, "perimetro_mm": 112.5, "masa_kg_m": 7.907},
4640     "Barra No.14": {"diametro_pulg": "1-3/4", "diametro_mm": 43.0, "area_mm2":
1452, "perimetro_mm": 135.1, "masa_kg_m": 11.38},
4641     "Barra No.18": {"diametro_pulg": "2-1/4", "diametro_mm": 57.3, "area_mm2":
2581, "perimetro_mm": 180.1, "masa_kg_m": 20.24}
4642 }
4643
4644 # Variables globales
4645 ventana_refuerzo_abierta = False
4646 ventana_refuerzo = None
4647 combo_barra = None
4648
4649
4650 # Declarar etiquetas globales para la información de la barra
4651 label_info_barra = None
4652 entry_Ab = None
4653
4654 def actualizar_info_barra():
4655     global combo_barra, label_info_barra, entry_Ab
4656     global entry_A , entry_B
4657     global label_barra, label_info_b, label_area_barra, label_0barras
4658
4659     barra_seleccionada = combo_barra.get()
4660
4661
4662     if barra_seleccionada != "Seleccione una barra":
4663
4664         info_barra = barras_info_refuerzo[barra_seleccionada]
4665
4666
4667         info_texto = (
4668             f"Barra No.{barra_seleccionada[9:]}\n\n"
4669             f"Diámetro: {info_barra['diametro_pulg']} pulgadas
({info_barra['diametro_mm']} mm)\n"
4670             f"Área: {info_barra['area_mm2']} mm²\n"
4671             f"Perímetro: {info_barra['perimetro_mm']} mm\n"
4672             f"Masa: {info_barra['masa_kg_m']} kg/m"
4673         )
4674
4675
4676         label_info_barra.config(text=info_texto)
4677
4678         area_barra = info_barra['area_mm2']
4679
4680         # Actualizar el label con el número de barra
4681         label_barra.config(text=f"No.{barra_seleccionada[9:]}")
4682
4683         # Actualizar el label con la información de la barra
4684         label_info_b.config(text=info_texto)
4685
4686         # Actualizar el área de la barra en el label_area_barra
4687         label_area_barra.config(text=f"{area_barra} mm²")
4688

```

```

4689     # Actualizar el diámetro en el label_Øbarras
4690     diametro_mm = info_barra['diametro_mm']
4691     label_Øbarras.config(text=f"{diametro_mm} mm")
4692
4693     if entry_Ab is not None:
4694         entry_Ab.delete(0, tk.END)
4695         entry_Ab.insert(0, str(area_barra))
4696
4697     # Llamar a la función para actualizar los cálculos del número de barras
4698     calcular_no_barras_refuerzo()
4699     # Llamar a la función para actualizar entry_A
4700     if entry_A is not None: # Asegúrate de que entry_A esté inicializado
4701         calcular_base_requerida(entry_A) # Pasar entry_A a la función
4702     else:
4703         print("Error: entry_A no está inicializado correctamente.")
4704
4705
4706 def calcular_no_barras_refuerzo():
4707     global entry_Ab, combo_As, combo_No_barras, combo_C, combo_barra, entry_D,
4708     combo_n, combo_dp, entry_A, entry_B, combo_b
4709     global Barras_traccion, label_barra_flexion,
4710     label_barra_flexion1, label_A_1, label_C_1, label_D_1, label_n_1, label_dp_1
4711     global label_base_req, label_base_req1, label_No_barras_flexion
4712     # Obtener la lista actualizada de valores de combo_As
4713     As_items = combo_As['values']
4714
4715     try:
4716         As_values = [float(item) for item in As_items if item.strip()]
4717     except ValueError:
4718         As_values = []
4719
4720     try:
4721         Ab = float(entry_Ab.get())
4722     except ValueError:
4723         Ab = 0
4724
4725     if Ab == 0 or not As_values:
4726         combo_No_barras['values'] = []
4727         combo_No_barras.set("")
4728         combo_C['values'] = []
4729         combo_C.set("")
4730         entry_D.delete(0, tk.END)
4731         entry_D.insert(0, "N/A")
4732         combo_n['values'] = []
4733         combo_n.set("")
4734         combo_dp['values'] = []
4735         combo_dp.set("")
4736         combo_b['values'] = []
4737         combo_b.set("")
4738         Barras_traccion['values'] = []
4739         Barras_traccion.set("")
4740         label_barra_flexion.config(text="")
4741         label_barra_flexion1.config(text="")
4742         label_A_1.config(text="")
4743         label_C_1.config(text="")
4744         label_D_1.config(text="")
4745         label_n_1.config(text="")
4746         label_dp_1.config(text="")
4747         label_base_req.config(text="")
4748         label_base_req1.config(text="")

```

```

4747     label_No_barras_flexion.config(text="")
4748     return
4749
4750     barra_seleccionada = combo_barra.get()
4751     barra_num = barra_seleccionada.split("No.")[1].strip() if "No." in
barra_seleccionada else "N/A"
4752
4753     # Actualizar el combobox combo_No_barras y Barras_traccion
4754     if combo_No_barras.get(): # Si hay algo seleccionado en combo_No_barras
4755         # Asignar los mismos valores de combo_No_barras a Barras_traccion
4756         Barras_traccion['values'] = combo_No_barras['values']
4757         Barras_traccion.set(combo_No_barras.get()) # Seleccionar el mismo valor
en Barras_traccion
4758     else:
4759         Barras_traccion['values'] = [] # Limpiar valores si no hay selección
4760
4761     # Actualizar los labels con los primeros dos valores de la lista de
combo_No_barras
4762     no_barras_values = combo_No_barras['values']
4763
4764     if no_barras_values:
4765         # Asignar el primer valor de la lista a label_barra_flexion
4766         label_barra_flexion.config(text=f"As[-]:{no_barras_values[0]}")
4767         label_No_barras_flexion.config(text=f"{no_barras_values[0]}") #
Actualización aquí
4768
4769         # Asignar el segundo valor de la lista a label_barra_flexion1 (si existe)
4770         if len(no_barras_values) > 1:
4771             label_barra_flexion1.config(text=f"As[+]:{no_barras_values[1]}")
4772         else:
4773             label_barra_flexion1.config(text="") # Si no hay segundo valor,
dejarlo vacío
4774
4775
4776     def redondear(val):
4777         if abs(val - 0) < 0.2 or abs(val - 1.0) < 0.2:
4778             return 2
4779         elif 0.1 < val < 1.0:
4780             return 2
4781         elif val % 1 > 0:
4782             return math.ceil(val)
4783         return val
4784
4785     def calcular_separacion_minima(numero_barra):
4786         """
4787         Calcula la separación mínima según las condiciones:
4788         - Para barras #2 a #8: 25 mm.
4789         - Para barras mayores a #8: diámetro de la barra.
4790         """
4791         try:
4792             diametro_mm = obtener_diametro_mm(numero_barra)
4793             if 2 <= numero_barra <= 8:
4794                 return 25 # Separación mínima fija
4795             elif numero_barra > 8:
4796                 return diametro_mm # Igual al diámetro de la barra
4797             else:
4798                 return 0 # Valor predeterminado en caso de error
4799         except Exception as e:
4800             print(f"Error calculando separación mínima: {e}")
4801             return 0

```

```

4802
4803
4804     No_barras_refuerzo_values = [redondear(As / Ab) for As in As_values]
4805     # Asegurar que el valor entero se muestre correctamente
4806     resultado_formateado = [f"{int(value)}#{barra_num}" if isinstance(value,
float) and value.is_integer() else f"{value}#{barra_num}" for value, barra_num in
zip(No_barras_refuerzo_values, [barra_num] * len(No_barras_refuerzo_values))]
4807
4808     combo_No_barras['values'] = resultado_formateado
4809     if resultado_formateado:
4810         combo_No_barras.set(resultado_formateado[0])
4811         # Agregar "mm" al formato de los valores en combo_C
4812         combo_C['values'] = [f"{calcular_radio_doblado(int(entry.split('#')[-1]))}
mm" for entry in resultado_formateado]
4813         combo_C.set(combo_C['values'][0])
4814
4815         # Calcular la separación mínima para la primera barra y mostrarla en
entry_D
4816         try:
4817             separacion_minima = calcular_separacion_minima(int(barra_num))
4818             entry_D.delete(0, tk.END)
4819             entry_D.insert(0, f"{separacion_minima} mm")
4820         except ValueError:
4821             entry_D.delete(0, tk.END)
4822             entry_D.insert(0, "N/A")
4823
4824         # Extraer el primer número de cada entrada en combo_No_barras y actualizar
combo_n
4825         numeros_primer_digito = [entry.split('#')[0] for entry in
resultado_formateado]
4826         combo_n['values'] = numeros_primer_digito
4827         if numeros_primer_digito:
4828             combo_n.set(numeros_primer_digito[0])
4829
4830         # Calcular y actualizar los diámetros de las barras longitudinales en
combo_dp
4831         diámetros_barras = [
4832             f"{obtener diametro_mm(int(entry.split('#')[-1]))} mm" for entry in
resultado_formateado
4833         ]
4834         combo_dp['values'] = diámetros_barras
4835         if diámetros_barras:
4836             combo_dp.set(diámetros_barras[0])
4837
4838         try:
4839             # Validar y obtener A
4840             entry_a_val = entry_A.get().replace(" mm", "").strip()
4841             A = float(entry_a_val) if entry_a_val else 0
4842
4843             # Validar y obtener B
4844             entry_b_val = entry_B.get().replace(" mm", "").strip()
4845             B = float(entry_b_val) if entry_b_val else 0
4846
4847             # Validar y obtener C
4848             combo_c_val = combo_C.get().replace(" mm", "").strip()
4849             C = float(combo_c_val) if combo_c_val else 0
4850
4851             # Validar y obtener D
4852             entry_d_val = entry_D.get().replace(" mm", "").strip()
4853             D = float(entry_d_val) if entry_d_val else 0

```

```

4854
4855 # Obtener dp del primer valor del combo_dp
4856 dp_val_clean = combo_dp.get().replace(" mm", "").strip()
4857 dp = float(dp_val_clean) if dp_val_clean else 0
4858
4859 # Actualizar los labels con los nuevos valores
4860 label_A_1.config(text=f"{A} mm")
4861 label_C_1.config(text=f"{C} mm")
4862 label_D_1.config(text=f"{D} mm")
4863 label_n_1.config(text=f"{numeros_primer_digito[0] if numeros_primer_digito
else 'N/A'}")
4864 label_dp_1.config(text=f"{dp} mm")
4865
4866 # Lista para almacenar resultados de base requerida (uno por cada n)
4867 base_requerida_vals = []
4868
4869 # Iterar sobre los valores en combo_n
4870 for combo_n_val in combo_n['values']:
4871     try:
4872         # Obtener el valor actual de n
4873         n = int(combo_n_val.strip()) if combo_n_val.strip().isdigit() else
1
4874
4875         # Calcular la base requerida
4876         base_requerida = 2 * (A + B + C) + (n - 1) * (D + dp)
4877         base_requerida_m = base_requerida / 1000 # Convertir a metros
4878
4879         # Agregar el resultado formateado a la lista
4880         base_requerida_vals.append(f"{base_requerida_m:.2f} m")
4881     except ValueError as n_error:
4882         pass
4883
4884 # Actualizar el combo_b con la lista calculada
4885 if base_requerida_vals:
4886     combo_b['values'] = base_requerida_vals
4887     combo_b.set(base_requerida_vals[0])
4888 else:
4889     combo_b['values'] = ["N/A"]
4890     combo_b.set("N/A")
4891
4892 # Actualizar los labels de base requerida
4893 if base_requerida_vals:
4894     label_base_req.config(text=f"{base_requerida_vals[0]}")
4895     if len(base_requerida_vals) > 1:
4896         label_base_req1.config(text=f"{base_requerida_vals[1]}")
4897     else:
4898         label_base_req1.config(text="") # Si no hay segundo valor,
dejarlo vacío
4899 else:
4900     label_base_req.config(text="N/A")
4901     label_base_req1.config(text="N/A")
4902 except ValueError as e:
4903     combo_b['values'] = ["N/A"]
4904     combo_b.set("N/A")
4905     label_base_req.config(text="N/A")
4906     label_base_req1.config(text="N/A")
4907
4908
4909 def calcular_radio_doblado(numero_barra):
4910     try:

```

```
4911     # Asignar un valor por defecto
4912     radio = 0
4913
4914     # Obtener el diámetro de la barra
4915     diametro_mm = obtener_diametro_mm(numero_barra)
4916
4917     # Validación de que se encontró el diámetro
4918     if diametro_mm == 0:
4919         raise ValueError("No se encontró información para el número de barra
proporcionado")
4920
4921     # Aplicar las condiciones para el cálculo
4922     if 2 <= numero_barra <= 5:
4923         # Para barras de #2 a #5: el radio es dos veces el diámetro
4924         radio = 2 * diametro_mm
4925     elif 6 <= numero_barra <= 8:
4926         # Para barras de #6 a #8: el radio es tres veces el diámetro
4927         radio = 3 * diametro_mm
4928     elif numero_barra > 8:
4929         # Para barras mayores a #8: el radio es igual al diámetro
4930         radio = diametro_mm
4931     else:
4932         # Caso seguro por si el número de barra es inválido
4933         radio = 0
4934     except Exception as e:
4935         print(f"Error calculando radio: {e}")
4936         radio = 0 # Asegurar un valor seguro en caso de error
4937
4938     return round(radio, 2) # Redondear el resultado a dos decimales
4939
4940 def obtener_diametro_mm(num_barra):
4941     """
4942     Devuelve el diámetro en mm basado en el número de barra.
4943     """
4944     clave = f"Barra No.{num_barra}"
4945     if clave in barras_info_refuerzo:
4946         return barras_info_refuerzo[clave]["diametro_mm"]
4947     else:
4948         print(f"No se encontró información para: {clave}")
4949     return 0
4950
4951
4952 def insertar_todos_los_datos_treeview():
4953     # Obtener los valores seleccionados de los combobox (solo los números)
4954     valores_mu_refuerzo = [valor.split()[0] for valor in Momento_Mu['values']] #
Obtener solo los números (eliminamos "kNm")
4955     valores_combo_as = combo_As['values'] # Obtener todos los valores disponibles
en combo_As
4956     valores_combo_no_barras = combo_No_barras['values'] # Obtener todos los
valores disponibles en combo_No_barras
4957     valores_combo_b = combo_b['values'] # Obtener todos los valores disponibles
en combo_b
4958
4959     # Limpiar las filas del Treeview antes de agregar nuevos datos
4960     for item in treeview_refuerzo.get_children():
4961         treeview_refuerzo.delete(item)
4962
4963     # Asegurarse de que el número de filas en todas las listas coincida o
completar con vacío si no es así
```

```

4964     max_len = max(len(valores_mu_refuerzo), len(valores_combo_as),
len(valores_combo_no_barras), len(valores_combo_b)) # Tomamos el máximo de las
listas para evitar errores
4965
4966     # Insertar los datos en el Treeview
4967     for i in range(max_len):
4968         # Si no hay suficientes valores en combo_As, usar un valor vacío
4969         combo_as_value = valores_combo_as[i] if i < len(valores_combo_as) else ""
4970
4971         # Si no hay suficientes valores en combo_No_barras, usar un valor vacío
4972         combo_no_barras_value = valores_combo_no_barras[i] if i <
len(valores_combo_no_barras) else ""
4973
4974         # Si no hay suficientes valores en valores_mu_refuerzo, usar un valor
vacío
4975         mu_refuerzo_value = valores_mu_refuerzo[i] if i < len(valores_mu_refuerzo)
else ""
4976
4977         # Si no hay suficientes valores en combo_b, usar un valor vacío
4978         combo_b_value = valores_combo_b[i] if i < len(valores_combo_b) else ""
4979
4980         # Insertar los valores en el Treeview (columna 1 para Momento_Mu, columna
2 para combo_As, columna 3 para combo_No_barras y columna 4 para combo_b)
4981         treeview_refuerzo.insert("", "end", values=(mu_refuerzo_value,
combo_as_value, combo_no_barras_value, combo_b_value))
4982
4983
4984         try:
4985             base_diseño = float(entry_prueba_base.get())
4986
4987         except ValueError:
4988             base_diseño = 0
4989
4990
4991
4992         if combo_b_value.strip() == "":
4993             base_requerida = 0
4994
4995         else:
4996
4997             base_requerida_str = re.sub(r"^[^d.]", "", combo_b_value)
4998             try:
4999                 base_requerida = float(base_requerida_str)
5000
5001             except ValueError:
5002                 base_requerida = 0
5003
5004
5005         # Comparación de las bases y actualización de los labels con los iconos
5006         if base_requerida < base_diseño:
5007             label_check_As.config(
5008                 text=f"La base requerida es menor al del diseño {base_diseño} m,
entonces cumple. ",
5009             )
5010             label_check_info.config(
5011                 text="Ya asignaste los aceros para cada Mu, ahora continúa con los
de compresión."
5012             )
5013             label_icon.config(image=icon_check_on)
5014

```

```

5015         else:
5016             label_check_As.config(
5017                 text=f"La base requerida es mayor al del diseño de {base_diseño}
m. Por lo tanto, no cumple.",
5018             )
5019             label_icon.config(image=icon_check_off)
5020
5021
5022
5023
5024 def abrir_ventana_barras_refuerzo():
5025     global ventana_refuerzo_abierta, ventana_refuerzo, combo_barra
5026     global label_info_barra, Barras_refuerzo, entry_Ab, combo_As, combo_No_barras,
entry_A
5027     global entry_B , combo_C , entry_D , combo_n, combo_dp,
combo_b,label_No_barras_flexion
5028
5029
5030     # Verificar si la ventana ya está abierta
5031     if not ventana_refuerzo_abierta:
5032         # Marcar que la ventana está abierta
5033         ventana_refuerzo_abierta = True
5034
5035         # Crear la nueva ventana
5036         ventana_refuerzo = tk.Toplevel()
5037         ventana_refuerzo.title("Barras de refuerzo")
5038         ventana_refuerzo.geometry("750x480")
5039         ventana_refuerzo.resizable(False, False)
5040         ventana_refuerzo.attributes('-topmost', True)
5041
5042         # Crear el LabelFrame principal
5043         labelframe_refuerzo = tk.LabelFrame(ventana_refuerzo, text="Barras de
refuerzo")
5044         labelframe_refuerzo.place(x=20, y=20, width=360, height=400)
5045
5046         # Etiqueta para pedir la selección de la barra
5047         label_refuerzo = tk.Label(labelframe_refuerzo, text="Seleccione la
designación de la barra:")
5048         label_refuerzo.place(x=70, y=10)
5049
5050         # Crear el combobox con las barras
5051         valores_barras = [
5052             "Barra No.2", "Barra No.3", "Barra No.4", "Barra No.5", "Barra No.6",
5053             "Barra No.7", "Barra No.8", "Barra No.9", "Barra No.10", "Barra
No.11",
5054             "Barra No.14", "Barra No.18"
5055         ]
5056
5057         combo_barra = ttk.Combobox(labelframe_refuerzo, values=["Seleccione una
barra"] + valores_barras)
5058         combo_barra.place(x=100, y=40)
5059         combo_barra.set("Seleccione una barra")
5060
5061         # Crear el LabelFrame para mostrar la información de la barra seleccionada
5062         labelframe_barras = tk.LabelFrame(labelframe_refuerzo, text="Información
de la barra")
5063         labelframe_barras.place(x=50, y=70, width=250, height=170)
5064
5065         # Crear una etiqueta para mostrar toda la información formateada de la
barra

```

```

5066         label_info_barra = tk.Label(labelframe_barras, text="", anchor="w",
justify="left")
5067         label_info_barra.pack(fill="both", padx=5, pady=15)
5068
5069         label_calculo_barras = tk.Label(labelframe_refuerzo, text="Cálculo número
de barras longitudinal: ")
5070         label_calculo_barras.place(x=70, y=250)
5071
5072         label_As_barras = tk.Label(labelframe_refuerzo, text="As: ")
5073         label_As_barras.place(x=80, y=280)
5074
5075         label_Ab_barras = tk.Label(labelframe_refuerzo, text="Ab: ")
5076         label_Ab_barras.place(x=80, y=310)
5077
5078         label_No_barras = tk.Label(labelframe_refuerzo, text="#barras: ")
5079         label_No_barras.place(x=70, y=340)
5080
5081         # Crear combobox As, Ab , # barras
5082         combo_As = ttk.Combobox(labelframe_refuerzo)
5083         combo_As.place(x=125, y=280, width=120)
5084         combo_As.set("Seleccionar")
5085
5086         entry_Ab = tk.Entry(labelframe_refuerzo)
5087         entry_Ab.place(x=125, y=310, width=120)
5088
5089         combo_No_barras = ttk.Combobox(labelframe_refuerzo)
5090         combo_No_barras.place(x=125, y=340, width=120)
5091
5092         label_No_barras_flexion =
tk.Label(labelframe_refuerzo, text="", bd=2, relief="sunken")
5093         label_No_barras_flexion.place(x=250, y=340, width=100)
5094
5095         labelframe_requisito = tk.LabelFrame(ventana_refuerzo, text="Requisito
mínimo ACI")
5096         labelframe_requisito.place(x=390, y=20, width=340, height=400)
5097
5098         label_requisito = tk.Label(labelframe_requisito, text="Revisar b=(ancho,
base sección transversal)")
5099         label_requisito.place(x=50, y=10)
5100
5101         label_formula = tk.Label(labelframe_requisito, text="b= 2[A+B+C]+[n-1]
[D+dp]", bd=2, relief="sunken")
5102         label_formula.place(x=90, y=35)
5103
5104         labelframe_formula = tk.LabelFrame(labelframe_requisito, text="")
5105         labelframe_formula.place(x=20, y=70, width=295, height=170)
5106
5107         label_A = tk.Label(labelframe_requisito, text="A:")
5108         label_A.place(x=10, y=260)
5109         entry_A = tk.Entry(labelframe_requisito)
5110         entry_A.place(x=40, y=260, width=80)
5111
5112         label_B = tk.Label(labelframe_requisito, text="B:")
5113         label_B.place(x=10, y=290)
5114         entry_B = tk.Entry(labelframe_requisito)
5115         entry_B.place(x=40, y=290, width=80)
5116
5117         label_C = tk.Label(labelframe_requisito, text="C:")
5118         label_C.place(x=10, y=320)
5119         combo_C = ttk.Combobox(labelframe_requisito)

```

```

5120     combo_C.place(x=40,y=320,width=80)
5121
5122     label_D = tk.Label(labelframe_requisito,text="D:")
5123     label_D.place(x=190,y=260)
5124     entry_D = tk.Entry(labelframe_requisito)
5125     entry_D.place(x=220,y=260,width=80)
5126
5127     label_n = tk.Label(labelframe_requisito,text="n:")
5128     label_n.place(x=190,y=290)
5129     combo_n = ttk.Combobox(labelframe_requisito)
5130     combo_n.place(x=220,y=290,width=80)
5131
5132     label_dp = tk.Label(labelframe_requisito,text="dp:")
5133     label_dp.place(x=190,y=320)
5134     combo_dp = ttk.Combobox(labelframe_requisito)
5135     combo_dp.place(x=220,y=320,width=80)
5136
5137     combo_b = ttk.Combobox(labelframe_requisito)
5138     #combo_b.place(x=120,y=370,width=80)
5139
5140
5141     # Crear el Canvas dentro de labelframe_formula
5142     Canvas_refuerzo = tk.Canvas(labelframe_formula,width=287, height=147)
5143     Canvas_refuerzo.place(x=0, y=0)
5144
5145     # Crear las barras de desplazamiento
5146     scrollbar_vertical = tk.Scrollbar(labelframe_formula, orient="vertical",
command=Canvas_refuerzo.yview)
5147     scrollbar_vertical.place(x=273, y=0, height=147)
5148
5149     scrollbar_horizontal = tk.Scrollbar(labelframe_formula,
orient="horizontal", command=Canvas_refuerzo.xview)
5150     scrollbar_horizontal.place(x=0, y=147, width=290)
5151
5152     # Vincular las scrollbars al Canvas
5153     Canvas_refuerzo.config(yscrollcommand=scrollbar_vertical.set,
xscrollcommand=scrollbar_horizontal.set)
5154
5155     # Crear la información que se mostrará en el Canvas
5156     text_refuerzo = """"b = 2[A + B + C] + [n - 1][D + dp]
5157     • A = 40mm mínimo para estribos (ACI 20.5.1)
5158     • B = diámetro del estribo: 9.5mm para No. 3 // 12.7mm para No. 4
5159     • C = radio de doblado (ACI Tabla 25.3.2),
5160     2db (para No. 3 a 5) o 3db (para No. 6 a 8)
5161     • D = espaciamiento mínimo (ACI 25.2.1): 25mm, db.
5162     - Nota 1: D=25mm para barras No.8 y menores / D=db para barras No.9 y
mayores)
5163     - Nota 2: Cuando hay diferentes diámetros, D = 1/2db de la barra mayor +
1/2db de la barra menor.
5164     • n = # de barras por refuerzo longitudinal
5165     • dp = Diámetro de las barras de refuerzo longitudinal""""
5166
5167     # Crear un contenedor de texto en el Canvas y obtener el tamaño del texto
5168     text_id = Canvas_refuerzo.create_text(5, 5, anchor="nw",
text=text_refuerzo, fill="black")
5169     # Función para actualizar el área de desplazamiento del Canvas
5170     def update_scrollregion(event=None):
5171         bbox = Canvas_refuerzo.bbox("all") # Obtener el área ocupada por el
contenido del Canvas
5172         if bbox:

```

```
5173         # Añadir un margen adicional de 100 píxeles a las coordenadas del área
5174         Canvas_refuerzo.config(scrollregion=(0, 0, bbox[2] + 100, bbox[3]
+ 30))
5175
5176         # Actualizar el área de desplazamiento después de crear el texto
5177         Canvas_refuerzo.bind("<Configure>", update_scrollregion)
5178
5179
5180         Canvas_refuerzo.config(scrollregion=Canvas_refuerzo.bbox("all"))
5181
5182
5183         agregar_lista = ttk.Button(ventana_refuerzo, text="Agregar
datos", command=insertar_todos_los_datos_treeview)
5184         agregar_lista.place(x=230, y=435, width=100)
5185
5186         aceptar_refuerzo=
5187         ttk.Button(ventana_refuerzo, text="Aceptar", command=cerrar_ventana_barras)
5188         aceptar_refuerzo.place(x=400, y=435, width=100)
5189
5190         if Barras_refuerzo is not None:
5191             combo_As['values'] = Barras_refuerzo['values']
5192
5193
5194         combo_barra.bind("<<ComboboxSelected>>", lambda event:
actualizar_info_barra())
5195
5196
5197         ventana_refuerzo.protocol("WM_DELETE_WINDOW", cerrar_ventana_barras)
5198
5199
5200     # Función para calcular la base requerida
5201     def calcular_base_requerida(entry_A):
5202         if entry_A is None:
5203             print("Error: entry_A no está definido")
5204             return
5205
5206
5207         valor_prueba_recubrimiento = entry_prueba_recubrimiento.get()
5208
5209
5210         match = re.match(r"([\d\.\.]+)", valor_prueba_recubrimiento)
5211         if match:
5212             valor_numerico = float(match.group(1))
5213         else:
5214             valor_numerico = 0
5215
5216
5217         entry_A.delete(0, tk.END)
5218         entry_A.insert(0, valor_prueba_recubrimiento)
5219
5220
5221         valor_label_estribo = label_prueba_estribo.cget("text")
5222
5223
5224         if entry_B is not None:
5225             entry_B.delete(0, tk.END)
5226             entry_B.insert(0, valor_label_estribo)
5227
5228
```

```
5229
5230
5231 # Variables globales
5232 ventana_mcr = None
5233 ventana_mcr_abierta = False
5234 entry_base_viga = None
5235 entry_altura_viga = None
5236 entry_recubrimiento_viga = None
5237 entry_d_viga = None
5238 canvas_seccion_1 = None
5239 combobox3 = None
5240 canvas_seccion_2 = None
5241 entry_As_prima = None
5242 entry_Ab_area = None
5243 entry_num_barra = None
5244 spinbox_barras = None
5245 label_mensaje3 = None
5246
5247 def actualizar_cotas(event=None):
5248     global entry_base_viga, entry_altura_viga, texto_ancho_id,
5249     texto_altura_id, canvas_seccion_1, combobox3
5250
5251     # Obtener los valores de las entradas
5252     try:
5253         ancho_base = float(entry_base_viga.get())
5254         altura_peralte = float(entry_altura_viga.get())
5255     except ValueError:
5256         # Si la entrada no es válida, limpiar el canvas
5257         canvas_seccion_1.delete("all")
5258         return # Salir si hay un error
5259
5260     # Limpiar el canvas antes de dibujar
5261     canvas_seccion_1.delete("all")
5262
5263     # Redibujar la figura
5264     if combobox3.get() == "Sección rectangular":
5265         dibujar_rectangulo(ancho_base, altura_peralte)
5266     elif combobox3.get() == "Sección cuadrada":
5267         dibujar_cuadrado(ancho_base)
5268
5269 def dibujar_rectangulo(ancho_base, altura_peralte):
5270     global figura_id, texto_ancho_id, texto_altura_id, canvas_seccion_1
5271
5272     # Escalar dimensiones
5273     ancho_base_px = ancho_base * 180
5274     altura_peralte_px = altura_peralte * 180
5275
5276     # Calcular coordenadas para centrar el rectángulo
5277     x1 = (130 - ancho_base_px) / 2
5278     y1 = (150 - altura_peralte_px) / 2
5279     x2 = x1 + ancho_base_px
5280     y2 = y1 + altura_peralte_px
5281
5282     # Dibujar el rectángulo y guardar su ID
5283     figura_id = canvas_seccion_1.create_rectangle(x1, y1, x2, y2, fill='gray',
5284     outline='black')
5285
5286     # Cotas del ancho (parte inferior) de canvas_seccion
5287     line_y = y2 + 8
5288     canvas_seccion_1.create_line(x1, line_y, x2, line_y, fill='black')
```

```

5287 canvas_seccion_1.create_line(x1, line_y, x1 + 5, line_y - 5, fill='black')
5288 canvas_seccion_1.create_line(x1, line_y, x1 + 5, line_y + 5, fill='black')
5289 canvas_seccion_1.create_line(x2, line_y, x2 - 5, line_y - 5, fill='black')
5290 canvas_seccion_1.create_line(x2, line_y, x2 - 5, line_y + 5, fill='black')
5291 texto_ancho_id = canvas_seccion_1.create_text((x1 + x2) / 2, line_y + 5,
text=f"{ancho_base} m", anchor='n')
5292
5293 # Cotas de la altura (parte lateral) de canvas_seccion
5294 line_x = x2 + 8
5295 canvas_seccion_1.create_line(line_x, y1, line_x, y2, fill='black')
5296 canvas_seccion_1.create_line(line_x, y1, line_x - 5, y1 + 5, fill='black')
5297 canvas_seccion_1.create_line(line_x, y1, line_x + 5, y1 + 5, fill='black')
5298 canvas_seccion_1.create_line(line_x, y2, line_x - 5, y2 - 5, fill='black')
5299 canvas_seccion_1.create_line(line_x, y2, line_x + 5, y2 - 5, fill='black')
5300 texto_altura_id = canvas_seccion_1.create_text(line_x + 10, (y1 + y2) / 2 +
15, text=f"{altura_peralte} m", anchor='w', angle=90)
5301
5302 def dibujar_cuadrado(tamano):
5303     global figura_id, texto_ancho_id, texto_altura_id, canvas_seccion_1
5304
5305     # Escalar dimensión
5306     tamano_px = tamano * 180
5307
5308     # Calcular coordenadas para centrar el cuadrado
5309     x1 = (130 - tamano_px) / 2
5310     y1 = (150 - tamano_px) / 2
5311     x2 = x1 + tamano_px
5312     y2 = y1 + tamano_px
5313
5314     # Dibujar el cuadrado y guardar su ID
5315     figura_id = canvas_seccion_1.create_rectangle(x1, y1, x2, y2, fill='gray',
outline='black')
5316
5317     # Cotas del ancho (parte inferior) de canvas_seccion
5318     line_y = y2 + 8
5319     canvas_seccion_1.create_line(x1, line_y, x2, line_y, fill='black')
5320     canvas_seccion_1.create_line(x1, line_y, x1 + 5, line_y - 5, fill='black')
5321     canvas_seccion_1.create_line(x1, line_y, x1 + 5, line_y + 5, fill='black')
5322     canvas_seccion_1.create_line(x2, line_y, x2 - 5, line_y - 5, fill='black')
5323     canvas_seccion_1.create_line(x2, line_y, x2 - 5, line_y + 5, fill='black')
5324     texto_ancho_id = canvas_seccion_1.create_text((x1 + x2) / 2, line_y + 5,
text=f"{tamano} m", anchor='n')
5325
5326     # Cotas de la altura (parte lateral) de canvas_seccion
5327     line_x = x2 + 8
5328     canvas_seccion_1.create_line(line_x, y1, line_x, y2, fill='black')
5329     canvas_seccion_1.create_line(line_x, y1, line_x - 5, y1 + 5, fill='black')
5330     canvas_seccion_1.create_line(line_x, y1, line_x + 5, y1 + 5, fill='black')
5331     canvas_seccion_1.create_line(line_x, y2, line_x - 5, y2 - 5, fill='black')
5332     canvas_seccion_1.create_line(line_x, y2, line_x + 5, y2 - 5, fill='black')
5333     texto_altura_id = canvas_seccion_1.create_text(line_x + 10, (y1 + y2) / 2 +
20, text=f"{tamano} m", anchor='w', angle=90)
5334
5335
5336 # Función para calcular d automáticamente
5337 def calcular_d(event=None):
5338     global entry_recubrimiento_viga, entry_altura_viga,
entry_d_viga, label_profundidad_prueba
5339     try:
5340         # Obtener los valores de las entradas

```

```

5341     altura_viga = float(entry_altura_viga.get())
5342     recubrimiento_viga = float(entry_recubrimiento_viga.get())
5343
5344     # Calcular la altura efectiva
5345     efectiva_d = altura_viga - recubrimiento_viga
5346
5347     # Actualizar el Entry de d
5348     entry_d_viga.delete(0, tk.END)
5349     entry_d_viga.insert(0, f"{efectiva_d:.2f}")
5350
5351     # Actualizar el texto del label
5352     label_profundidad_prueba.config(text=f"{efectiva_d:.2f}")
5353
5354     except ValueError:
5355         entry_d_viga.delete(0, tk.END)
5356         label_profundidad_prueba.config(text="Valor inválido")
5357
5358 # Función para calcular Mcr
5359 def calcular_Mcr():
5360     global label_mensaje1, entry_base_viga, entry_altura_viga, combobox1,
5361     entry_Mcr, label_Mcr_1, label_Mcr_check
5362     try:
5363         # Obtener el valor de f'c desde el combobox, quitar "MPa" y convertirlo a
5364         float
5365         f_c_str = combobox1.get().replace(" MPa", "") # Remover " MPa"
5366         f_c = float(f_c_str)
5367
5368         # Obtener el valor de la base y la altura desde los entries y convertirlos
5369         a float
5370         base_Mcr = float(entry_base_viga.get())
5371         altura_Mcr = float(entry_altura_viga.get())
5372
5373         # Calcular Mcr usando las variables base_Mcr y altura_Mcr
5374         Mcr = 0.62 * sqrt(f_c) * base_Mcr * (altura_Mcr ** 2) * 1000 / 6
5375
5376         # Mostrar el resultado en el entry de Mcr
5377         entry_Mcr.delete(0, tk.END)
5378         entry_Mcr.insert(0, f"{Mcr:.2f}") # Insertar el resultado formateado a
5379         dos decimales
5380
5381         # Actualizar el mensaje en el label
5382         label_mensaje1.config(text=(
5383             f"Momento de agrietamiento: {Mcr:.2f} kNm\n"
5384             "Indica el límite donde el concreto comienza a agrietarse."),
5385             fg="black")
5386
5387         # Actualizar el dato en label_Mcr_1
5388         label_Mcr_1.config(text=f"{Mcr:.2f} kNm.")
5389
5390         # Actualizar el texto en label_Mcr_check con el mismo mensaje de
5391         label_mensaje1
5392         label_Mcr_check.config(
5393             text=(f"Momento de agrietamiento: {Mcr:.2f} kNm.\n"
5394                 "Indica el límite donde el concreto comienza a agrietarse."),
5395             fg="black"
5396         )
5397
5398     except ValueError:
5399         entry_Mcr.delete(0, tk.END)
5400         entry_Mcr.insert(0, "Error: Verifique los valores")

```

```

5395     error_msg = "Error: Verifique los valores"
5396     label_mensaje1.config(text="Error: Verifique los valores", fg="red")
5397     # Actualizar los labels en caso de error
5398     label_Mcr_1.config(text="Error en los datos de entrada", fg="red")
5399     label_Mcr_check.config(text=error_msg, fg="red")
5400 except Exception as e:
5401     entry_Mcr.delete(0, tk.END)
5402     entry_Mcr.insert(0, f"Error: {str(e)}")
5403     label_mensaje1.config(text=f"Error: {str(e)}", fg="red")
5404     # Actualizar los labels en caso de error
5405     label_Mcr_1.config(text=f"Error: {str(e)}", fg="red")
5406     label_Mcr_check.config(text=error_msg, fg="red")
5407
5408
5409 # Función para calcular la cuantía requerida (ρ)
5410 def Calcular_Cuantia_Req(event=None):
5411     try:
5412         # Obtener los valores seleccionados de los Combobox y el Entry
5413         # Extraer el número de f'_c desde el Combobox1
5414         f_c_1 = float(combobox1.get().split()[0])
5415         f_y_1 = float(combobox2.get().split()[0])
5416         Rn_2 = float(entry_Rn.get()) / 1000
5417
5418         # Constante 0.85
5419         constante_c = 0.85
5420
5421         # Calcular ρ usando la fórmula proporcionada
5422         raiz_1 = 1 - (2 * Rn_2) / (constante_c * f_c_1)
5423
5424         if raiz_1 < 0:
5425             # Si el valor bajo la raíz cuadrada es negativo, mostramos el error
5426             entry_p.config(state="normal")
5427             entry_p.delete(0, tk.END)
5428             entry_p.insert(0, "Error en cálculo")
5429             entry_p.config(state="readonly")
5430             # Actualizar label_preq_mcr con mensaje de error
5431             label_preq_mcr.config(text="Error en cálculo")
5432             # Limpiar label_Rn_mcr_check
5433             label_Rn_mcr_check.config(text="")
5434         else:
5435             # Si todo es válido, calculamos la cuantía (ρ)
5436             rho_4 = (constante_c * f_c_1 / f_y_1) * (1 - math.sqrt(raiz_1))
5437
5438             # Mostrar el resultado de ρ en el Entry (de solo lectura)
5439             entry_p.config(state="normal")
5440             entry_p.delete(0, tk.END)
5441             entry_p.insert(0, str(round(rho_4, 6)))
5442             entry_p.config(state="readonly")
5443
5444             # Actualizar label_preq_mcr con el valor de ρ calculado
5445             label_preq_mcr.config(text=f"{round(rho_4, 6)}")
5446
5447             # Comparar con la cuantía mínima
5448             cuantia_minima = 0.0033
5449             if rho_4 < cuantia_minima:
5450                 # Si ρ es menor que la cuantía mínima, mostrar el mensaje
5451                 mensaje_mcr = (f"La cuantía requerida es {rho_4:.5f}. "
5452                               f"Es menor que la cuantía mínima de
5453                               {cuantia_minima:.5f}. "
5454                               "Se debe usar la cuantía mínima.")

```

```

5454         label_mensaje2.config(text=mensaje_mcr)
5455         # Actualizar label_Rn_mcr_check con el mismo mensaje de
label_mensaje2
5456         label_Rn_mcr_check.config(text=mensaje_mcr, bg="white")
5457     else:
5458         # Si p es mayor o igual a la cuantía mínima, mostrar un mensaje
diferente
5459         mensaje_mcr = (f"La cuantía requerida es {rho_4:.5f}. "
5460                        "Es adecuada para el diseño.")
5461         label_mensaje2.config(text=mensaje_mcr)
5462         # Actualizar label_Rn_mcr_check con el mismo mensaje de
label_mensaje2
5463         label_Rn_mcr_check.config(text=mensaje_mcr, bg="white")
5464
5465     except ValueError:
5466         # Si ocurre un error en la conversión de los datos o cualquier otro
problema
5467         entry_p.config(state="normal")
5468         entry_p.delete(0, tk.END)
5469         entry_p.insert(0, "Error en datos")
5470         entry_p.config(state="readonly")
5471
5472         # Limpiar los otros labels
5473         label_preq_mcr.config(text="Error en los datos", bg="red")
5474         label_Rn_mcr_check.config(text="", bg="white")
5475         label_mensaje2.config(text="")
5476         label_mensaje2.config(text="")
5477
5478
5479 def Calcular_Resistencia_Rn():
5480     try:
5481         # Obtener los valores de los campos de entrada con los nuevos nombres de las
variables
5482         Mcr_momento_1 = float(entry_Mcr.get()) # Obtener el valor de Mcr
5483         b_base_1 = float(entry_base_viga.get()) # Obtener el valor de b
5484         d_altura_efectiva_1 = float(entry_d_viga.get()) # Obtener el valor de d
5485
5486         # Constante Phi_0.90_1
5487         Phi_1 = 0.90
5488
5489         # Calcular Rn usando la fórmula  $Rn = Mcr / (\Phi * b * d^2)$ 
5490         Rn_1 = Mcr_momento_1 / (Phi_1 * b_base_1 * d_altura_efectiva_1 ** 2)
5491
5492         # Mostrar el resultado en entry_Rn
5493         entry_Rn.delete(0, tk.END) # Limpiar el entry antes de poner el nuevo
valor
5494         entry_Rn.insert(0, str(round(Rn_1, 2))) # Insertar el valor de Rn
(redondeado a 2 decimales)
5495
5496         # Actualizar el texto del label_b_prueba con el valor de entry_base_viga
5497         label_b_prueba.config(text=f"{b_base_1}") # Actualizar el texto del label
5498
5499         # Actualizar el valor de label_Rn_mcr con el mismo valor calculado
5500         label_Rn_mcr.config(text=f"{round(Rn_1, 2)} kN/m²") # Mostrar el valor de
Rn en el label
5501
5502     except ValueError:
5503         # Si no se pueden convertir los valores a float, mostrar un mensaje de error
5504         entry_Rn.delete(0, tk.END)
5505         entry_Rn.insert(0, "Error en los datos")

```

```
5506         # Actualizar label_Rn_mcr con el mensaje de error
5507         label_Rn_mcr.config(text="Error en los datos")
5508
5509     def on_checkbutton_Rn_select():
5510         Calcular_Resistencia_Rn()
5511         Calcular_Cuantia_Req()
5512
5513
5514     # Diccionario con la información de las barras de refuerzo
5515     barras_info_refuerzo_1= {
5516         "Barra No.2": {"diametro_pulg": "1/4", "diametro_mm": 6.4, "area_mm2": 32,
5517         "perimetro_mm": 20.0, "masa_kg_m": 0.250},
5518         "Barra No.3": {"diametro_pulg": "3/8", "diametro_mm": 9.5, "area_mm2": 71,
5519         "perimetro_mm": 30.0, "masa_kg_m": 0.560},
5520         "Barra No.4": {"diametro_pulg": "1/2", "diametro_mm": 12.7, "area_mm2": 129,
5521         "perimetro_mm": 40.0, "masa_kg_m": 0.994},
5522         "Barra No.5": {"diametro_pulg": "5/8", "diametro_mm": 15.9, "area_mm2": 199,
5523         "perimetro_mm": 50.0, "masa_kg_m": 1.552},
5524         "Barra No.6": {"diametro_pulg": "3/4", "diametro_mm": 19.1, "area_mm2": 284,
5525         "perimetro_mm": 60.0, "masa_kg_m": 2.235},
5526         "Barra No.7": {"diametro_pulg": "7/8", "diametro_mm": 22.2, "area_mm2": 387,
5527         "perimetro_mm": 70.0, "masa_kg_m": 3.042},
5528         "Barra No.8": {"diametro_pulg": "1", "diametro_mm": 25.4, "area_mm2": 510,
5529         "perimetro_mm": 80.0, "masa_kg_m": 3.973},
5530         "Barra No.9": {"diametro_pulg": "1-1/8", "diametro_mm": 28.7, "area_mm2": 645,
5531         "perimetro_mm": 90.0, "masa_kg_m": 5.060},
5532         "Barra No.10": {"diametro_pulg": "1-1/4", "diametro_mm": 32.3, "area_mm2": 819,
5533         "perimetro_mm": 101.3, "masa_kg_m": 6.404},
5534         "Barra No.11": {"diametro_pulg": "1-3/8", "diametro_mm": 35.8, "area_mm2": 1006,
5535         "perimetro_mm": 112.5, "masa_kg_m": 7.907},
5536         "Barra No.14": {"diametro_pulg": "1-3/4", "diametro_mm": 43.0, "area_mm2": 1452,
5537         "perimetro_mm": 135.1, "masa_kg_m": 11.38},
5538         "Barra No.18": {"diametro_pulg": "2-1/4", "diametro_mm": 57.3, "area_mm2": 2581,
5539         "perimetro_mm": 180.1, "masa_kg_m": 20.24}
5540     }
5541
5542     # Función combinada para actualizar información y realizar cálculos
5543     def actualizar_y_calcular(event=None):
5544         actualizar_label_informacion()
5545         calculos_previos()
5546         actualizar_dibujar()
5547
5548     # Función para obtener información de una barra específica
5549     def obtener_informacion_barra(seleccion_2):
5550         barra_info = barras_info_refuerzo_1.get(seleccion_2)
5551         if barra_info:
5552             return (
5553                 f"{seleccion_2}:\n"
5554                 f"Diámetro: {barra_info['diametro_pulg']} pulgadas\n"
5555                 f"({barra_info['diametro_mm']} mm)\n"
5556                 f"Área: {barra_info['area_mm2']} mm²\n"
5557                 f"Perímetro: {barra_info['perimetro_mm']} mm\n"
5558                 f"Masa: {barra_info['masa_kg_m']} kg/m"
5559             )
5560         else:
5561             return "Designación de barra no encontrada."
```

```
5553 # Función para mostrar información de la barra seleccionada
5554 def actualizar_label_informacion(event=None):
5555     seleccion_2 = spinbox_barras.get()
5556     informacion = obtener_informacion_barra(seleccion_2)
5557     label_mensaje3.config(text=informacion)
5558
5559
5560
5561 # Función para realizar cálculos
5562 def calculos_previos(event=None):
5563     try:
5564         b_base = float(entry_base_viga.get())
5565         d_peralte = float(entry_d_viga.get())
5566     except ValueError:
5567         return
5568
5569     seleccion_2 = spinbox_barras.get()
5570     barra_info = barras_info_refuerzo_1.get(seleccion_2)
5571     if barra_info is None:
5572         return
5573
5574     As_compresión_1 = 0.0033 * b_base * 1000 * d_peralte * 1000
5575     entry_As_prima.delete(0, tk.END)
5576     entry_As_prima.insert(0, f"{As_compresión_1:.2f}")
5577
5578     # Actualizar el label correspondiente para As'
5579     label_area_req_mcr.config(text=f"{As_compresión_1:.2f} mm²")
5580
5581     area_barra = barra_info["area_mm2"]
5582     entry_Ab_area.delete(0, tk.END)
5583     entry_Ab_area.insert(0, f"{area_barra}")
5584
5585     # Actualizar el label correspondiente para el área de barra
5586     label_area_barra_mcr.config(text=f"{area_barra} mm²")
5587
5588     try:
5589         As_prima = float(entry_As_prima.get())
5590         Ab_area = float(entry_Ab_area.get())
5591         numero_de_barras = As_prima / Ab_area
5592
5593         numero_de_barras_redondeado = (
5594             math.floor(numero_de_barras)
5595             if numero_de_barras - math.floor(numero_de_barras) < 0.90
5596             else math.ceil(numero_de_barras)
5597         )
5598
5599         if numero_de_barras_redondeado < 2:
5600             numero_de_barras_redondeado = 2
5601
5602         barra_numero_1 = re.search(r"\d+", seleccion_2).group()
5603         entry_num_barra.delete(0, tk.END)
5604         entry_num_barra.insert(0, f"{numero_de_barras_redondeado}#{barra_numero_1}")
5605
5606         # Actualizar el label correspondiente para el número de barras
5607         label_num_barra_mcr.config(text=f"{numero_de_barras_redondeado}#{barra_numero_1}")
5608
5609         # Actualizar el Combobox con el mismo valor
5610         Barras_compresion.set(f"{numero_de_barras_redondeado}#{barra_numero_1}")
```

```
5611
5612     # Ahora, actualizar el label_info_barras_mcr con la información completa
de la barra
5613     barra_info_texto = obtener_informacion_barra(seleccion_2)
5614     label_info_barras_mcr.config(text=barra_info_texto)
5615
5616     except ValueError:
5617         pass
5618
5619
5620 # Función para dibujar la figura en el Canvas
5621 def actualizar_dibujar(event=None):
5622     # Obtener los valores de base y altura de los entrys
5623     try:
5624         base_2 = float(entry_base_viga.get())
5625         altura_2 = float(entry_altura_viga.get())
5626     except ValueError:
5627         return # Si no se ingresan valores válidos, no hacer nada
5628
5629     # Obtener el número de círculos desde el entry entry_num_barra
5630     num_circulos_texto = entry_num_barra.get()
5631     try:
5632         num_circulos = int(num_circulos_texto.split('#')[0])
5633     except ValueError:
5634         num_circulos = 0
5635
5636     # Factor de escala para el tamaño de la figura
5637     escala = 200
5638     offset_interno = 10
5639     radio_circulo = 3
5640
5641     # Limpiar el canvas antes de redibujar
5642     canvas_seccion_2.delete("all")
5643
5644     # Calcular el tamaño escalado de la figura
5645     base_escalada = base_2 * escala
5646     altura_escalada = altura_2 * escala
5647
5648     # Calcular las coordenadas para centrar la figura
5649     canvas_width_1 = canvas_seccion_2.winfo_width()
5650     canvas_height_1 = canvas_seccion_2.winfo_height()
5651
5652     # Coordenadas para centrar la figura en el Canvas
5653     x_offset = (canvas_width_1 - base_escalada) / 2
5654     y_offset = (canvas_height_1 - altura_escalada) / 2
5655
5656     # Crear el texto a mostrar arriba de la figura externa
5657     base_viga_cm = int(base_2 * 100)
5658     altura_viga_cm = int(altura_2 * 100)
5659     texto_viga = f"VG {base_viga_cm} X {altura_viga_cm}"
5660
5661     # Calcular tamaño de fuente proporcional a la figura externa
5662     tamano_fuente = int(base_escalada / 7)
5663
5664     # Dibujar el texto sobre la figura externa en el Canvas
5665     canvas_seccion_2.create_text(
5666         x_offset + base_escalada / 2,
5667         y_offset - tamano_fuente - 5,
5668         text=texto_viga,
5669         font=("Arial", tamano_fuente, "bold"),
```

```

5670         fill="black"
5671     )
5672
5673     # Dibujar la figura externa (original) en el canvas
5674     canvas_seccion_2.create_rectangle(
5675         x_offset, y_offset,
5676         x_offset + base_escalada, y_offset + altura_escalada,
5677         outline="black", fill="lightgrey", width=2
5678     )
5679
5680     # Dibujar la figura interna con el offset aplicado
5681     base_interna = base_escalada - 2 * offset_interno
5682     altura_interna = altura_escalada - 2 * offset_interno
5683
5684     canvas_seccion_2.create_rectangle(
5685         x_offset + offset_interno, y_offset + offset_interno,
5686         x_offset + base_escalada - offset_interno, y_offset + altura_escalada -
5687         offset_interno,
5688         outline="red", width=2
5689     )
5690
5691     # Dibujar círculos dentro de la figura interna en la parte superior
5692     if num_circulos > 0:
5693         # Espaciado entre los círculos
5694         espacio_lateral = (base_interna - (num_circulos - 1) * (2 * radio_circulo
5695         + 10) - 2 * radio_circulo) / 2
5696
5697         for i in range(num_circulos):
5698             # Coordenadas x para cada círculo
5699             if i == 0:
5700                 x_circulo = x_offset + offset_interno + radio_circulo
5701             elif i == num_circulos - 1:
5702                 x_circulo = x_offset + base_escalada - offset_interno -
5703                 radio_circulo
5704             else:
5705                 x_circulo = x_offset + offset_interno + espacio_lateral + i * (2 *
5706                 radio_circulo + 10) + radio_circulo
5707
5708             # Coordenada y (ubicada cerca de la parte superior de la figura
5709             interna)
5710             y_circulo = y_offset + offset_interno + radio_circulo
5711
5712             # Dibujar el círculo en el canvas
5713             canvas_seccion_2.create_oval(
5714                 x_circulo - radio_circulo, y_circulo - radio_circulo,
5715                 x_circulo + radio_circulo, y_circulo + radio_circulo,
5716                 outline="black", fill="black", width=2
5717             )
5718
5719     # Dibujar las cotas (etiquetas de ancho y altura)
5720
5721     # Cota de ancho (base)
5722     canvas_seccion_2.create_line(
5723         x_offset-10, y_offset + altura_escalada + 10,
5724         x_offset + base_escalada + 10, y_offset + altura_escalada + 10,
5725         arrow=tk.BOTH, width=2
5726     )
5727
5728     canvas_seccion_2.create_text(
5729         (x_offset + x_offset + base_escalada) / 2,
5730         y_offset + altura_escalada + 20,

```

```

5725     text=f"{base_2} m", font=("Arial", 10, "bold")
5726 )
5727
5728 # Ajuste dinámico para la cota de altura basado en el tamaño de la base
5729 ajuste_lateral_cota = base_escalada * 1.15
5730
5731 # Cota de altura
5732 canvas_seccion_2.create_line(
5733     x_offset + ajuste_lateral_cota, y_offset - 10,
5734     x_offset + ajuste_lateral_cota, y_offset + altura_escalada + 10,
5735     arrow=tk.BOTH, width=2
5736 )
5737 canvas_seccion_2.create_text(
5738     x_offset + ajuste_lateral_cota + 10,
5739     (y_offset + y_offset + altura_escalada) / 2,
5740     text=f"{altura_2} m", font=("Arial", 10, "bold"), angle=90
5741 )
5742
5743 # Función para actualizar el texto en el Canvas
5744 def actualizar_texto_canvas():
5745
5746     canvas_seccion_2.delete("texto_canvas")
5747
5748     if check_var.get():
5749         texto = entry_num_barra.get()
5750
5751         texto_formateado = f"As': {texto}"
5752         canvas_seccion_2.create_text(85, 85, text=texto_formateado,
tags="texto_canvas", font=("Arial", 7, "bold"), fill="black")
5753
5754
5755 def on_entry_change_1(*args):
5756     if check_var.get():
5757         actualizar_texto_canvas()
5758
5759
5760 # Función para insertar datos en el Treeview
5761 def insertar_datos_treeview_3():
5762     # Obtener los valores de los cuadros de texto
5763     viga_1 = entry_seccion_viga_VG.get()
5764     mcr = entry_Mcr.get()
5765     pmin_1 = entry_pmin.get()
5766     as_prima = entry_As_prima.get()
5767     num_barra_1 = entry_num_barra.get()
5768
5769     # Insertar datos en la primera fila del Treeview
5770     treeview_Mcr.delete(*treeview_Mcr.get_children()) # Limpiar el Treeview
5771     treeview_Mcr.insert("", "end", values=(viga_1, mcr, pmin_1, as_prima,
num_barra_1))
5772
5773     # Actualizar el texto del Label
5774     mensaje_Mcr = (
5775         f"La viga con sección {viga_1} requiere un área de acero de {as_prima} mm²
"
5776         f"en la zona de compresión equivalente a {num_barra_1}."
5777     )
5778
5779     mensaje_desarrollo = (
5780         'Muy bien, ahora, continúa con las configuraciones de longitud de
desarrollo.'
```

```

5781     )
5782
5783     label_check_Mcr.config(text=mensaje_Mcr)
5784     label_icon_Mcr.config(image=icon_check_on)
5785     label_check_desarrollo.config(text=mensaje_desarrollo)
5786
5787
5788 def cerrar_ventana_Mcr():
5789     global ventana_mcr, ventana_mcr_abierta
5790     ventana_mcr_abierta = False
5791     ventana_mcr.destroy()
5792     ventana_mcr = None
5793
5794 #Función para abrir ventana Momento agrietamiento
5795 def abrir_ventana_Mcr():
5796     global ventana_mcr, ventana_mcr_abierta,entry_base_viga, entry_altura_viga,
5797     canvas_seccion_1,combobox3
5798     global
5799     entry_recubrimiento_viga,entry_d_viga,label_mensaje1,entry_Mcr,combobox1,boton_Mcr
5800     global entry_Rn, entry_p, label_mensaje1, label_mensaje2,combobox2,check_var
5801     global
5802     canvas_seccion_2,entry_As_prima,entry_Ab_area,spinbox_barras,label_mensaje3,entry_
5803     num_barra,barras_info_refuerzo_1,valores_barras_Spin
5804     global entry_seccion_viga_VG,entry_pmin
5805     # Si la ventana ya está abierta, no crear otra
5806     if ventana_mcr_abierta:
5807         return
5808
5809     ventana_mcr = tk.Toplevel()
5810     ventana_mcr_abierta = True
5811     ventana_mcr.title("Cálculo momento de agrietamiento")
5812     ventana_mcr.geometry("800x465")
5813     ventana_mcr.resizable(False, False)
5814     ventana_mcr.attributes('-topmost', True)
5815
5816     # Agregar contenido a la ventana Mcr
5817     etiqueta_calculos = tk.LabelFrame(ventana_mcr, text="Datos de la viga")
5818     etiqueta_calculos.place(x=10, y=10, width=380, height=100)
5819     etiqueta_sección_viga = tk.LabelFrame(ventana_mcr, text="Sección transversal")
5820     etiqueta_sección_viga.place(x=10, y=110, width=380, height=200)
5821     etiqueta_momento_viga = tk.LabelFrame(ventana_mcr, text="Momento de
5822     agrietamiento")
5823     etiqueta_momento_viga.place(x=10, y=310, width=380, height=100)
5824     etiqueta_Rn_viga = tk.LabelFrame(ventana_mcr, text="Resistencia Requerida y
5825     Cuantia Requerida")
5826     etiqueta_Rn_viga.place(x=400, y=10, width=390, height=120)
5827     etiqueta_As_viga = tk.LabelFrame(ventana_mcr, text="Asignar Refuerzo a
5828     Compresión")
5829     etiqueta_As_viga.place(x=400,y=130, width=390, height=280)
5830
5831     # Crear un Canvas dentro del LabelFrame
5832     canvas_seccion_1 = tk.Canvas(etiqueta_sección_viga,
5833     bg='white',bd=4,relief="ridge" ,width=130, height=150)
5834     canvas_seccion_1.place(x=220, y=10)
5835     canvas_seccion_2 = tk.Canvas(etiqueta_As_viga, bg='white',bd=4,relief="ridge"
5836     ,width=160, height=170)
5837     canvas_seccion_2.place(x=210, y=40)
5838
5839     # Agregar label a la ventana Mcr

```

```

5832 label_seccion_VG = tk.Label(etiqueta_calculos, text="Sección:")
5833 label_seccion_VG.place(x=10, y=10)
5834 label_seccion_luz_VG = tk.Label(etiqueta_calculos, text="Luz:")
5835 label_seccion_luz_VG.place(x=10, y=40)
5836 label_material_fc= tk.Label(etiqueta_calculos,text="f'c:")
5837 label_material_fc.place(x=200, y=10)
5838 label_material_fy= tk.Label(etiqueta_calculos,text="fy:")
5839 label_material_fy.place(x=200, y=40)
5840 label_tipo_viga_VG = tk.Label(etiqueta_sección_viga, text="Tipo:")
5841 label_tipo_viga_VG.place(x=10,y=10)
5842 label_base_viga = tk.Label(etiqueta_sección_viga, text="b:")
5843 label_base_viga.place(x=10,y=50)
5844 label_unidad_base = tk.Label(etiqueta_sección_viga, text="m")
5845 label_unidad_base.place(x=80,y=50)
5846 label_altura_viga = tk.Label(etiqueta_sección_viga, text="h:")
5847 label_altura_viga.place(x=120,y=50)
5848 label_unidad_altura = tk.Label(etiqueta_sección_viga, text="m")
5849 label_unidad_altura.place(x=190,y=50)
5850 label_recubrimiento_viga = tk.Label(etiqueta_sección_viga, text="r:")
5851 label_recubrimiento_viga.place(x=10,y=90)
5852 label_unidad_recubrimiento = tk.Label(etiqueta_sección_viga, text="m")
5853 label_unidad_recubrimiento.place(x=80,y=90)
5854 label_d_viga = tk.Label(etiqueta_sección_viga, text="d:")
5855 label_d_viga.place(x=120,y=90)
5856 label_unidad_d = tk.Label(etiqueta_sección_viga, text="m")
5857 label_unidad_d.place(x=190,y=90)
5858 label_Mcr = tk.Label(etiqueta_momento_viga, text="Mcr:")
5859 label_Mcr.place(x=70,y=20)
5860 label_unidad_Mcr = tk.Label(etiqueta_momento_viga, text="kNm")
5861 label_unidad_Mcr.place(x=200,y=20)
5862 label_mensaje1= tk.Label(etiqueta_momento_viga,text="")
5863 label_mensaje1.place(x=10,y=45)
5864 label_Rn =tk.Label(etiqueta_Rn_viga,text="Rn:")
5865 label_Rn.place(x=6,y=40)
5866 label_unidad_Rn=tk.Label(etiqueta_Rn_viga,text="kN/m²")
5867 label_unidad_Rn.place(x=115,y=40)
5868 label_p=tk.Label(etiqueta_Rn_viga,text="preg:")
5869 label_p.place(x=250,y=40)
5870 label_pmin = tk.Label(etiqueta_Rn_viga,text="pmin:")
5871 label_pmin.place(x=250,y=6)
5872 label_mensaje2 = tk.Label(etiqueta_Rn_viga, text="", wraplength=350,
justify="left", anchor="w", width=50)
5873 label_mensaje2.place(x=1,y=60)
5874 label_escoger_barra =tk.Label(etiqueta_As_viga,text="Seleccione la designación
de la barra:")
5875 label_escoger_barra.place(x=6,y=6)
5876 label_mensaje3 = tk.Label(etiqueta_As_viga,text="",justify="left")
5877 label_mensaje3.place(x=10, y=50, width=200, height=110)
5878 label_As_prima = tk.Label(etiqueta_As_viga,text="As'")
5879 label_As_prima.place(x=10,y=160)
5880 label_Ab_area = tk.Label(etiqueta_As_viga,text="Ab:")
5881 label_Ab_area.place(x=10,y=190)
5882 label_num_barra = tk.Label(etiqueta_As_viga,text="#barras:")
5883 label_num_barra.place(x=10,y=220)
5884 label_unidad_As=tk.Label(etiqueta_As_viga,text="mm²")
5885 label_unidad_As.place(x=140,y=160)
5886 label_unidad_Ab=tk.Label(etiqueta_As_viga,text="mm²")
5887 label_unidad_Ab.place(x=140,y=190)
5888
5889 # Agregar entrys en la ventana Mcr

```

```
5890 entry_seccion_viga_VG = tk.Entry(etiqueta_calculos)
5891 entry_seccion_viga_VG.place(x=80, y=10, width=80)
5892 entry_seccion_luz = tk.Entry(etiqueta_calculos)
5893 entry_seccion_luz.place (x=80,y=40, width=80 )
5894 entry_base_viga = tk.Entry(etiqueta_sección_viga)
5895 entry_base_viga.place(x=30, y=50 ,width=50)
5896 entry_altura_viga = tk.Entry(etiqueta_sección_viga)
5897 entry_altura_viga.place(x=140, y=50 ,width=50)
5898 entry_recubrimiento_viga = tk.Entry(etiqueta_sección_viga)
5899 entry_recubrimiento_viga.place(x=30, y=90 ,width=50)
5900 entry_d_viga =tk.Entry(etiqueta_sección_viga)
5901 entry_d_viga.place(x=140,y=90, width=50)
5902 entry_Mcr =tk.Entry(etiqueta_momento_viga)
5903 entry_Mcr.place(x=110,y=20,width=80)
5904 entry_Rn =tk.Entry(etiqueta_Rn_viga)
5905 entry_Rn.place(x=33,y=40,width=80)
5906 entry_p =tk.Entry(etiqueta_Rn_viga)
5907 entry_p.place(x=290,y=40,width=70)
5908 entry_pmin =tk.Entry(etiqueta_Rn_viga)
5909 entry_pmin.place(x=290,y=6,width=70)
5910 entry_pmin.insert(0, "0.0033")
5911 entry_pmin.config(state='readonly')
5912 entry_As_prima= tk.Entry(etiqueta_As_viga)
5913 entry_As_prima.place(x=65,y=160,width=70)
5914 entry_Ab_area= tk.Entry(etiqueta_As_viga)
5915 entry_Ab_area.place(x=65,y=190,width=70)
5916 entry_num_barra= tk.Entry(etiqueta_As_viga)
5917 entry_num_barra.place(x=65,y=220,width=70)
5918
5919 # Botón para guardar datos de entradas y cerrar ventana_calculos
5920 boton_guardar_datos_Mcr = ttk.Button(ventana_mcr,text="Agregar
datos",command=insertar_datos_treeview_3)
5921 boton_guardar_datos_Mcr.place(x=270,y=420,width=100)
5922 boton_aceptar_ventana =
ttk.Button(ventana_mcr,text="Aceptar",command=cerrar_ventana_Mcr)
5923 boton_aceptar_ventana.place(x=420,y=420,width=100)
5924
5925 # Asignar los valores de los Label a los Entry
5926 entry_seccion_viga_VG.insert(0, label_seccion_cargas.cget("text"))
5927 entry_seccion_luz.insert(0, label_seccion_luz.cget("text")+"m")
5928
5929 # Variable para almacenar el estado del Checkbutton
5930 check_var = tk.BooleanVar()
5931
5932 # Crear el Checkbutton en la interfaz
5933 check_button = tk.Checkbutton(etiqueta_As_viga, text="As'",
variable=check_var,command=actualizar_texto_canvas)
5934 check_button.place(x=140,y=220)
5935
5936 # Variable de control para el Checkbutton Rn
5937 var_Rn = tk.BooleanVar()
5938
5939 # Variable de control para otro Checkbutton
5940 var_seccion_constante = tk.BooleanVar()
5941
5942 # Crear el Checkbutton
5943 checkbutton_Mcr = tk.Checkbutton(etiqueta_sección_viga, text="Mantener sección
constante",variable=var_seccion_constante)
5944 checkbutton_Mcr.place(x=10,y=130)
5945
```

```
5946 # Crear el Checkbutton
5947 Checkbutton_Rn = tk.Checkbutton(etiqueta_Rn_viga, text="Considerar  $\phi = 0,90$ 
para tracción", variable=var_Rn, command=on_checkbutton_Rn_select)
5948 Checkbutton_Rn.place(x=6, y=6)
5949
5950 # Crear el botón para calcular Mcr
5951 boton_Mcr = ttk.Button(etiqueta_momento_viga, text="Calcular Mcr",
command=calcular_Mcr)
5952 boton_Mcr.place(x=250, y=15)
5953
5954
5955
5956 # Agregar Combobox o lista de datos
5957 # Lista de opciones para el Combobox
5958 opciones_fc = [
5959 "17.5 MPa",
5960 "21 MPa",
5961 "24.5 MPa",
5962 "28 MPa",
5963 "35 MPa",
5964 "42 MPa"
5965 ]
5966
5967 opciones_fy = [
5968 "420 Mpa"
5969 ]
5970
5971 opciones2= [
5972 "Sección rectangular",
5973 "Sección cuadrada"
5974 ]
5975
5976 # Crear el Combobox
5977 combobox1 = ttk.Combobox(etiqueta_calculos, values=opciones_fc,
state="readonly")
5978 combobox1.set("Seleccionar") # Establecer el valor inicial
5979 combobox1.place(x=240, y=10, width=100)
5980
5981 combobox2 = ttk.Combobox(etiqueta_calculos, values=opciones_fy,
state="readonly")
5982 combobox2.set("Seleccionar") # Establecer el valor inicial
5983 combobox2.place(x=240, y=40, width=100)
5984
5985 combobox3 = ttk.Combobox(etiqueta_sección_viga, values=opciones2,
state="readonly")
5986 combobox3.set("Seleccionar") # Establecer el valor inicial
5987 combobox3.place(x=50, y=10, width=130)
5988
5989 # Crear el Spinbox usando la lista de valores
5990 spinbox_barras = ttk.Spinbox(etiqueta_As_viga,
values=list(barras_info_refuerzo_1.keys()))
5991 spinbox_barras.place(x=10, y=40)
5992 spinbox_barras.bind("<ButtonRelease-1>", actualizar_y_calcular)
5993
5994
5995 entry_base_viga.bind("<KeyRelease>", actualizar_cotas)
5996 entry_altura_viga.bind("<KeyRelease>", actualizar_cotas)
5997 entry_recubrimiento_viga.bind("<KeyRelease>", calcular_d)
5998
5999
```

```

6000     ventana_mcr.protocol("WM_DELETE_WINDOW", cerrar_ventana_Mcr)
6001
6002 # Globalmente almacenamos el objeto canvas
6003 canvas = None
6004
6005 def guardar_imagen(canvas):
6006     # Ruta relativa donde se guardará la imagen
6007     ruta_guardado_1 = "Reportes/imagenes_pilot/grafico_final.png"
6008
6009     # Guardar la figura con fondo transparente
6010     if canvas is not None:
6011         canvas.figure.savefig(ruta_guardado_1, dpi=300, transparent=True)
6012
6013 def Dibujar_Despiece_Viga():
6014     global canvas
6015     # Obtener el valor del Entry de la longitud de la viga L (eliminamos la parte
6016     " m")
6017     valor_cota = entry_luz_Ld.get() # Longitud de la viga L
6018     L_despiece = float(valor_cota.split()[0]) # Tomamos solo la parte numérica y
6019     la convertimos a flotante
6020
6021     # Obtener el valor de Wu desde el Entry
6022     valor_Wu = entry_Wu_Ld.get() # Wu
6023     Wu_ld = float(valor_Wu.split()[0]) # Tomamos solo la parte numérica y la
6024     convertimos a flotante
6025
6026     # Crear la figura y el eje
6027     fig, ax = plt.subplots(figsize=(8, 6))
6028
6029     # Definir las coordenadas de los puntos (x, y) para la figura original (sin la
6030     línea duplicada)
6031     puntos_viga_original = [
6032         (1, 0.5), (1, 4.5), # Línea vertical izquierda alargada
6033         (1, 4.5), (3, 4.5), # Línea horizontal superior izquierda
6034         (3, 4.5), (3, 3.5), # Línea vertical derecha (corta, baja a un tercio)
6035         (3, 3.5), (8, 3.5), # Línea horizontal proyectada más alargada desde el
6036         tercio
6037         (8, 3.5), (8, 4.5), # Línea vertical corta subiendo desde la línea
6038         horizontal proyectada
6039         (8, 4.5), (10, 4.5), # Línea horizontal superior derecha proyectada
6040         (10, 4.5), (10, 0.5), # Línea vertical derecha alargada
6041         (10, 0.5), (8, 0.5), # Línea horizontal inferior de la misma longitud
6042         (8, 0.5), (8, 1.5), # Línea corta vertical subiendo a un tercio
6043         (8, 1.5), (3, 1.5), # Línea horizontal proyectada desde el tercio hacia
6044         la izquierda
6045         (3, 1.5), (3, 0.5), # Línea corta vertical bajando a la misma longitud de
6046         la línea vertical anterior
6047         (3, 0.5), (1, 0.5), # Línea horizontal inferior en sentido izquierdo
6048     ]
6049
6050     # Definir las coordenadas de la línea duplicada como una cota
6051     linea_duplicada = [(3, 0.4), (8, 0.4)] # Línea horizontal separada desplazada
6052     hacia abajo
6053
6054     # Dibujar la figura original
6055     x_original = [p[0] for p in puntos_viga_original]
6056     y_original = [p[1] for p in puntos_viga_original]
6057     ax.fill(x_original, y_original, color='lightgrey', alpha=0.5) # Hacemos la
6058     figura translúcida para distinción
6059     ax.plot(x_original, y_original, 'k-', linewidth=2)

```

```
6050
6051     # Dibujar la cota
6052     x_linea_cota = [p[0] for p in linea_duplicada]
6053     y_linea_cota = [p[1] for p in linea_duplicada]
6054     ax.plot(x_linea_cota, y_linea_cota, 'r-', linewidth=1.5, label='Cota') #
Línea de cota
6055
6056     # Líneas de extensión
6057     ax.plot([3, 3], [0.4, 0.7], 'r--', linewidth=1) # Línea de extensión
izquierda
6058     ax.plot([8, 8], [0.4, 0.7], 'r--', linewidth=1) # Línea de extensión derecha
6059
6060     # Flechas en los extremos
6061     ax.annotate('', xy=(3, 0.4), xytext=(8, 0.4), arrowprops=dict(arrowstyle='<-
>', color='red', lw=1.5))
6062
6063     # Texto de la cota (valor del Entry)
6064     ax.text(5.5, 0.5, valor_cota, color='red', fontsize=8, ha='center',
fontweight='bold') # Ajustamos el texto dinámicamente
6065
6066     # Dibujar la figura copia (sin la cota), desplazándola hacia arriba
6067     x_copia = [p[0] for p in puntos_viga_original]
6068     y_copia = [p[1] + 5 for p in puntos_viga_original] # Desplazamos hacia arriba
6069     ax.fill(x_copia, y_copia, color='lightgrey', alpha=0.5)
6070     ax.plot(x_copia, y_copia, 'k--', linewidth=2)
6071
6072
6073     # Calcular el centro de la figura copia en el eje Y
6074     y_centro = (min(y_copia) + max(y_copia)) / 2
6075
6076     # Dibujar la línea horizontal negra continua
6077     ax.plot([3, 8], [y_centro, y_centro], 'k-', linewidth=1.5, label='Línea
centrada')
6078
6079     # Dibujar el primer punto rojo en la figura copia
6080     ax.plot(3, 4.5 + 5, 'ro')
6081
6082     # Dibujar el segundo punto rojo
6083     ax.plot(5.5, 1.5 + 5, 'ro')
6084
6085     # Dibujar el tercer punto rojo en la figura copia
6086     # Coordenadas de los tres puntos rojos
6087     puntos_rojos = np.array([(3, 4.5 + 5), (5.5, 1.5 + 5), (8, 4.5 + 5)])
6088
6089     # Ajustar los puntos para la curva
6090     x_puntos = puntos_rojos[:, 0]
6091     y_puntos = puntos_rojos[:, 1]
6092
6093     # Interpolación cúbica para obtener la curva que pasa por los puntos
6094     spline = CubicSpline(x_puntos, y_puntos, bc_type='natural')
6095
6096     # Crear puntos en el eje X para dibujar la curva
6097     x_dibujo = np.linspace(min(x_puntos), max(x_puntos), 500)
6098     y_dibujo = spline(x_dibujo)
6099
6100     # Dibujar la curva
6101     ax.plot(x_dibujo, y_dibujo, 'r-', label='Curva cóncava hacia abajo')
6102
6103     # Encontrar los puntos de intersección con la línea horizontal
6104     intersecciones_x = np.linspace(min(x_dibujo), max(x_dibujo), 500)
```

```

6105     intersecciones_y = spline(intersecciones_x)
6106
6107     # Suponiendo que la línea horizontal está en y_centro
6108     intersecciones = np.where(np.isclose(intersecciones_y, y_centro, atol=0.1))[0]
6109
6110     # Si hay intersecciones, encontrar la más a la izquierda
6111     if len(intersecciones) > 0:
6112         interseccion_x_izquierda = intersecciones_x[intersecciones[0]] # El
primer índice es el más a la izquierda
6113         interseccion_y_izquierda = intersecciones_y[intersecciones[0]]
6114
6115         # Intersección más a la derecha (mayor valor de x)
6116         interseccion_x_derecha = intersecciones_x[intersecciones[-1]]
6117         interseccion_y_derecha = intersecciones_y[intersecciones[-1]]
6118
6119         # Dibujar el punto rojo en la intersección más a la izquierda
6120         ax.plot(interseccion_x_izquierda, interseccion_y_izquierda, 'o',
color='red', markersize=6, label='Intersección con la línea horizontal')
6121
6122         # Dibujar el punto rojo en la intersección más a la derecha
6123         ax.plot(interseccion_x_derecha, interseccion_y_derecha, 'o', color='red',
markersize=6, label='Intersección derecha')
6124
6125         # Línea de extensión vertical para la intersección izquierda (hacia abajo)
6126         ax.plot([interseccion_x_izquierda, interseccion_x_izquierda],
[interseccion_y_izquierda, min(intersecciones_y) - 5], 'r--', linewidth=1.5)
6127
6128         # Línea de extensión vertical para la intersección derecha (hacia abajo)
6129         ax.plot([interseccion_x_derecha, interseccion_x_derecha],
[interseccion_y_derecha, min(intersecciones_y) - 5], 'r--', linewidth=1.5)
6130
6131         # Agregar texto a la línea de extensión izquierda (vertical)
6132         ax.text(interseccion_x_izquierda - 0.3, (interseccion_y_izquierda +
min(intersecciones_y)) / 2.8, 'P. Inflexión',
6133             color='red', fontsize=9, rotation=90, verticalalignment='center',
horizontalalignment='left', fontweight='bold')
6134
6135         # Agregar texto a la línea de extensión derecha (vertical)
6136         ax.text(interseccion_x_derecha + 0.1, (interseccion_y_derecha +
min(intersecciones_y)) / 2.8, 'P. Inflexión',
6137             color='red', fontsize=9, rotation=90, verticalalignment='center',
horizontalalignment='left', fontweight='bold')
6138
6139         # Calcular el punto medio entre las dos intersecciones
6140         interseccion_x_centro = (interseccion_x_izquierda + interseccion_x_derecha) /
2
6141         interseccion_y_centro = (interseccion_y_izquierda + interseccion_y_derecha) /
2
6142
6143         # Desplazar el punto central hacia abajo
6144         desplazamiento_y = abs(max(intersecciones_y) - min(intersecciones_y)) * 0.36
6145         interseccion_y_centro_desplazado = interseccion_y_centro - desplazamiento_y
6146
6147         # Dibujar un punto más abajo del centro
6148         ax.plot(interseccion_x_centro, interseccion_y_centro_desplazado, 'o',
color='red', markersize=6, label='Centro desplazado')
6149
6150         # Línea de extensión duplicada hacia otro punto (por ejemplo, desplazada en x
hacia la derecha)
6151         nuevo_punto_x = interseccion_x_centro + 2.5 # Desplazamos en el eje X

```

```

6152     nuevo_punto_y = interseccion_y_centro_desplazado # Mantenemos la misma altura
        inicial
6153
6154     # Desplazamos hacia la izquierda en el eje X
6155     nuevo_punto_x_izquierda = interseccion_x_centro - 2.5 # Desplazamos a la
        izquierda
6156     nuevo_punto_y_izquierda = interseccion_y_centro_desplazado # Mantenemos la
        misma altura inicial
6157
6158     # Dibujar la nueva línea de extensión desplazada hacia la izquierda
6159     ax.plot([nuevo_punto_x_izquierda, nuevo_punto_x_izquierda],
        [nuevo_punto_y_izquierda, min(intersecciones_y) + 2], 'g--', linewidth=1.5)
6160
6161     # Dibujar la nueva línea de extensión
6162     ax.plot([nuevo_punto_x, nuevo_punto_x], [nuevo_punto_y, min(intersecciones_y)
        + 2], 'g--', linewidth=1.5)
6163
6164     # Línea de extensión vertical desde el punto central desplazado hacia abajo
6165     ax.plot([interseccion_x_centro, interseccion_x_centro],
        [interseccion_y_centro_desplazado, min(intersecciones_y) + 2], 'g--',
        linewidth=1.5)
6166
6167     # Agregar texto al punto desplazado (horizontal, sin rotación)
6168     ax.text(interseccion_x_centro, interseccion_y_centro_desplazado +
        desplazamiento_y + 1.1, 'Sección Crítica',
6169         color='green', fontsize=9, verticalalignment='center',
        horizontalalignment='center', fontweight='bold')
6170
6171     # Colorear las áreas bajo la curva entre los puntos de intersección
6172     if len(intersecciones) >= 2:
6173         for i in range(len(intersecciones) - 1):
6174             x1, x2 = intersecciones_x[intersecciones[i]],
        intersecciones_x[intersecciones[i + 1]]
6175             y1, y2 = spline(x1), spline(x2)
6176             ax.fill_between(x_dibujo, y_dibujo, y_centro, where=(x_dibujo >= x1) &
        (x_dibujo <= x2), color='blue', alpha=0.5)
6177
6178     # Colorear la área antes de la primera intersección (de la línea horizontal
        hasta el primer punto de intersección)
6179     if len(intersecciones) > 0:
6180         x_inicio = 3 # El inicio de la línea horizontal
6181         x_fin_1 = intersecciones_x[intersecciones[0]]
6182         ax.fill_between(x_dibujo, y_dibujo, y_centro, where=(x_dibujo >= x_inicio)
        & (x_dibujo <= x_fin_1), color='lightcoral', alpha=1)
6183
6184     # Colorear la área después de la segunda intersección (desde el segundo punto
        de intersección hasta el final de la línea horizontal)
6185     if len(intersecciones) > 1:
6186         x_inicio_2 = intersecciones_x[intersecciones[-1]]
6187         x_fin_2 = 8 # El fin de la línea horizontal
6188         ax.fill_between(x_dibujo, y_dibujo, y_centro, where=(x_dibujo >=
        x_inicio_2) & (x_dibujo <= x_fin_2), color='lightcoral', alpha=1)
6189
6190
6191     # Calcular los momentos máximos negativo y positivo
6192     M1_ld = (1 / 12) * Wu_ld * L_despiece ** 2
6193     M2_ld = (1 / 24) * Wu_ld * L_despiece ** 2
6194     M3_ld = (1 / 12) * Wu_ld * L_despiece ** 2
6195
6196     # Colocar las etiquetas con los momentos

```

```

6197     ax.text(3, 4.5 + 5.2, f'Mu[-]: {M1_ld:.2f} kNm', color='black', fontsize=8,
ha='center', fontweight='bold')
6198     ax.text(5.5, 1.5 + 4.5, f'Mu[+]: {M2_ld:.2f} kNm', color='black', fontsize=8,
ha='center', fontweight='bold')
6199     ax.text(8, 4.5 + 5.2, f'Mu[-]: {M3_ld:.2f} kNm', color='black', fontsize=8,
ha='center', fontweight='bold')
6200
6201
6202     # Ajustar la visualización
6203     ax.set_xlim(0, 11)
6204     ax.set_ylim(-0.1, 10) # Escalamos los límites para acomodar ambas figuras
6205
6206     # Usar un aspecto automático y ajustar márgenes
6207     ax.set_aspect('auto') # Elimina la relación fija entre los ejes para que las
figuras no se reduzcan
6208     fig.tight_layout() # Ajusta automáticamente los márgenes de la figura
6209     # Ocultar los ejes
6210     ax.axis('off') # Esto elimina los ejes, marcas y etiquetas
6211
6212
6213     # Incrustar la figura en el frame de Tkinter
6214     canvas = FigureCanvasTkAgg(fig, master=frame_despiece) # 'frame_despiece' es
el contenedor
6215     canvas.draw() # Dibuja la figura
6216
6217     # Mostrar el gráfico usando place con la posición ajustada en la parte
superior
6218     canvas.get_tk_widget().place(x=-10, y=10, width=frame_despiece.winfo_width(),
height=frame_despiece.winfo_height() * 0.88)
6219
6220 def Dibujar_Despiece_Aceros_Flexion():
6221     global canvas
6222
6223     if canvas is not None:
6224         # Obtener el eje del canvas para agregar la nueva línea
6225         ax = canvas.figure.axes[0] # Usamos el primer eje de la figura ya
dibujada
6226
6227         # Obtener el texto actual de los labels (estos valores son tratados como
texto)
6228         As_negativo_izquierda_valor = label_As_negativo_izquierda.cget("text")
6229         As_positivo_centro_valor = label_As_positivo_centro.cget("text")
6230         As_negativo_derecha_valor = label_As_negativo_derecha.cget("text")
6231
6232         # Obtener el valor de Barras_compresion del Combobox
6233         Barras_compresion_valor = Barras_compresion.get()
6234
6235         # Obtener el texto del label de Tkinter
6236         texto_label_seccion_cargas = label_seccion_cargas.cget('text')
6237         texto_label_seccion_luz = "Luz: " + label_seccion_luz.cget('text') + " m"
6238
6239         # Definir las coordenadas de la nueva línea As_positivo
6240         As_positivo = [3.5, 7.5]
6241         y_linea_positiva = [1.7, 1.7]
6242
6243         # Dibujar la línea As[+] en azul
6244         ax.plot(As_positivo, y_linea_positiva, 'b-', linewidth=2) # Línea azul
6245
6246         # Agregar el texto con el valor de As_positivo_centro

```

```
6247     ax.text(5.5, 1.9, f'As[+]: {As_positivo_centro_valor}', fontsize=7,
ha='center', color='blue',fontweight='bold') # Coloca el texto justo sobre la
línea As[+]

6248
6249     # Definir las coordenadas de As_negativo
6250     As_negativo = [1.2, 4.8]
6251     y_linea_negativa = [3.3, 3.3]
6252
6253     # Dibujar la línea As[-] en roja
6254     ax.plot(As_negativo, y_linea_negativa, 'r-', linewidth=2) # Línea roja
6255
6256     # Agregar el texto con el valor de As_negativo_izquierda
6257     ax.text(2.2, 3.5, f'As[-]: {As_negativo_izquierda_valor}', fontsize=7,
ha='center', color='red',fontweight='bold') # Coloca el texto justo sobre la
línea As[-]

6258
6259     # Agregar una línea vertical roja en la coordenada x = 1.2
6260     y_vertical = [2.5, 3.3]
6261     ax.plot([1.2, 1.2], y_vertical, 'r-', linewidth=2) # Línea vertical roja
6262
6263     # Copiar la línea As[-] y moverla a la derecha
6264     As_negativo_copia = [x + 5 for x in As_negativo]
6265     ax.plot(As_negativo_copia, y_linea_negativa, 'r-', linewidth=2) # Copia
de As[-] en roja
6266
6267     # Agregar el texto con el valor de As_negativo_derecha
6268     ax.text(8.9, 3.5, f'As[-]: {As_negativo_derecha_valor}', fontsize=7,
ha='center', color='red',fontweight='bold') # Coloca el texto justo sobre la
línea copiada
6269
6270     # Copiar la línea vertical y moverla a la derecha
6271     x_vertical_copia = 1.2 + 8.6
6272     ax.plot([x_vertical_copia, x_vertical_copia], y_vertical, 'r-',
linewidth=2) # Copia de la línea vertical roja
6273
6274     # Definir las coordenadas de la línea As_compresión
6275     As_compresion = [1.3, 4.5]
6276     y_linea_compresion = [1.85, 1.85]
6277
6278     # Dibujar la línea As_compresión en color verde
6279     ax.plot(As_compresion, y_linea_compresion, 'g-', linewidth=2,
label='As_compresión') # Línea verde
6280
6281     # Agregar el texto con el valor de Barras_compresion
6282     ax.text(2.9, 1.9, f"As': {Barras_compresion_valor}", fontsize=7,
ha='center', color='green', fontweight='bold')
6283
6284     # Agregar una línea vertical verde en la coordenada x = 1.3
6285     y_vertical_verde = [1.85, 2.5] # Ajuste del rango en el eje y para la
línea vertical verde
6286     ax.plot([1.3, 1.3], y_vertical_verde, 'g-', linewidth=2) # Línea vertical
verde
6287
6288     # Hacer una copia de la línea As_compresión y moverla a la derecha
6289     As_compresion_copia = [x + 5.2 for x in As_compresion]
6290     ax.plot(As_compresion_copia, y_linea_compresion, 'g-', linewidth=2) #
Copia de la línea As_compresión en verde
6291
6292     ax.text(8, 1.9, f"As': {Barras_compresion_valor}", fontsize=7,
ha='center', color='green', fontweight='bold')
```

```

6293
6294     # Hacer una copia de la línea vertical verde y moverla a la derecha
6295     x_vertical_verde_copia = 1.3 + 8.4
6296     ax.plot([x_vertical_verde_copia, x_vertical_verde_copia],
y_vertical_verde, 'g-', linewidth=2) # Copia de la línea vertical verde
6297
6298     # Definir las coordenadas de la línea As_compresion_centro (más arriba de
la línea azul)
6299     As_compresion_centro = [3.7, 7.3] # Usamos las mismas coordenadas que
As_positivo
6300     y_linea_compresion_centro = [3.17, 3.17]
6301
6302     # Dibujar la nueva línea As_compresion_centro en color naranja
6303     ax.plot(As_compresion_centro, y_linea_compresion_centro, 'g-',
linewidth=2, label='As_compresion_centro') # Línea naranja
6304
6305     ax.text(5.5, 2.8, f"As': {Barras_compresion_valor}", fontsize=7,
ha='center', color='green', fontweight='bold')
6306
6307     # Crear una leyenda personalizada
6308     legend_elements = [
6309         Line2D([0], [0], color='blue', lw=2, label=f'As[+]:
{As_positivo_centro_valor}'),
6310         Line2D([0], [0], color='red', lw=2, label=f'As[-] Izq.:
{As_negativo_izquierda_valor}'),
6311         Line2D([0], [0], color='red', lw=2, label=f'As[-] Der.:
{As_negativo_derecha_valor}'),
6312         Line2D([0], [0], color='green', lw=2, label=f"As':
{Barras_compresion_valor}"),
6313         Line2D([0], [0], color='grey', lw=0, marker='s',
label=texto_label_seccion_cargas),
6314         Line2D([0], [0], color='black', lw=2, label=texto_label_seccion_luz)
6315     ]
6316
6317     # Crear una propiedad de fuente personalizada para el título
6318     bold_title_font = font_manager.FontProperties(weight='bold', size=7)
6319
6320     ax.legend(
6321         handles=legend_elements,
6322         loc='lower center', # Posición de la leyenda
6323         bbox_to_anchor=(0.5, -0.081), # Ajuste de posición
6324         fontsize=7,
6325         title='Leyenda',
6326         title_fontproperties=bold_title_font, # Negrita para el título
6327         frameon=True,
6328         prop={'size': 7, 'weight': 'bold'},
6329         handlelength=1.5,
6330         labelspacing=0.4,
6331         ncol=3,
6332         edgecolor='lightgrey',
6333         facecolor='white'
6334     )
6335
6336
6337
6338
6339     # Redibujar la figura con todas las líneas
6340     canvas.draw()
6341
6342     # Llamar a la función guardar_imagen después de dibujar

```

```

6343         guardar_imagen(canvas)
6344
6345     # Ejecutar ambas funciones
6346     Dibujar_Despiece_Aceros_Flexion()
6347
6348     # Variable global para controlar si la ventana ya está abierta
6349     ventana_Ld = None
6350
6351     label_As_negativo_izquierda = tk.Label(ventana,text="",bg="red")
6352     #label_As_negativo_izquierda.place(x=200,y=650,width=100)
6353     label_As_positivo_centro = tk.Label(ventana,text="",bg="blue")
6354     #label_As_positivo_centro.place(x=300,y=650,width=100)
6355     label_As_negativo_derecha = tk.Label(ventana,text="",bg="red")
6356     #label_As_negativo_derecha.place(x=400,y=650,width=100)
6357
6358     label_Ldh_traccion = tk.Label(ventana,text="",bg="green")
6359     #label_Ldh_traccion.place(x=500,y=650,width=100)
6360     label_Ldh_compresion = tk.Label(ventana,text="",bg="blue")
6361     #label_Ldh_compresion.place(x=600,y=650,width=100)
6362     label_Ldh_traccion_longitud = tk.Label(ventana,text="",bg="green")
6363     #label_Ldh_traccion_longitud.place(x=700,y=650,width=100)
6364     label_Ldh_compresion_longitud = tk.Label(ventana,text="",bg="blue")
6365     #label_Ldh_compresion_longitud.place(x=800,y=650,width=100)
6366
6367
6368     def abrir_ventana_longitud_desarrollo():
6369         global ventana_Ld,frame_despiece,
6370         entry_luz_Ld,entry_Wu_Ld,Barras_traccion,Momento_Mu,combo_barras_traccion
6371         global
6372         combo_ψt,combo_ψe,combo_ψs,combo_Cb,combo_λ,combo_db,combo_Ld,combo_datos_fc,boton
        _calcular_Ld
6373         global
6374         check_button_0,combo_barras_diametro,combo_barras_ldc,Barras_compresion,combo_barr
        as_diametro_ldc
6375         global
6376         combo_fc_Ldc,combo_Ldc,label_Ldc_db,label_Ldc_db_1,label_18db_1,label_resultados_L
        dc,combo_ganchos,frame_ganchos
6377         global
6378         combo_barras_doblado,label_diametro_doblado_1,label_concreto_λ_factor,label_factor
        _ψe_1,label_resultado_Ldh_1
6379         global
6380         combo_concreto_Ldh,treeview_Ldh,label_treeview_Ldh,treeview_Ldh_compresion,label_t
        reeview_Ldh_compresion
6381         global treeview_ganchos,label_treeview_ganchos,boton_agregar_datos,treeview_Ld
6382         if ventana_Ld is not None and ventana_Ld.winfo_exists():
6383             return
6384
6385         # Crear la ventana
6386         ventana_Ld = tk.Toplevel()
6387         ventana_Ld.title("Longitud de desarrollo")
6388         ventana_Ld.geometry("900x700")
6389         ventana_Ld.resizable(False, False)
6390         ventana_Ld.attributes('-topmost', True)
6391
6392         #Crear contenido en la ventana longitud desarrollo
6393         labelframe_Ld = tk.LabelFrame(ventana_Ld,text="")
6394         labelframe_Ld.place(x=20,y=20,width=860,height=50)
6395
6396         label_tipo_Ld = tk.Label(labelframe_Ld,text="Tipo viga:")
6397         label_tipo_Ld .place(x=10,y=10)

```

```

6392     entry_tipo_Ld = tk.Entry(labelframe_Ld)
6393     entry_tipo_Ld.place(x=80,y=10,width=110)
6394
6395
6396     label_seccion_Ld = tk.Label(labelframe_Ld,text="Sección:")
6397     label_seccion_Ld.place(x=200,y=10)
6398     entry_seccion_Ld = tk.Entry(labelframe_Ld)
6399     entry_seccion_Ld.place(x=260,y=10,width=110)
6400
6401
6402     label_luz_Ld= tk.Label(labelframe_Ld,text="Luz:")
6403     label_luz_Ld.place(x=380,y=10)
6404     entry_luz_Ld = tk.Entry(labelframe_Ld)
6405     entry_luz_Ld.place(x=420,y=10,width=100)
6406
6407     label_Wu_Ld = tk.Label(labelframe_Ld,text="Wu:")
6408     label_Wu_Ld.place(x=540,y=10)
6409     entry_Wu_Ld = tk.Entry(labelframe_Ld)
6410     entry_Wu_Ld.place(x=570,y=10,width=100)
6411
6412     boton_Ld=
6413     ttk.Button(labelframe_Ld,text="Aplicar",command=Dibujar_Despiece_Viga)
6414     boton_Ld.place(x=700,y=10,width=100)
6415
6416     labelframe_opciones = tk.LabelFrame(ventana_Ld,text="")
6417     labelframe_opciones.place(x=20,y=640,width=860,height=50)
6418
6419     boton_dibujar_flexion = ttk.Button(ventana_Ld,
6420     text="Dibujar",command=Dibujar_Despiece_Aceros_Flexion)
6421     boton_dibujar_flexion.place(x=200, y=650, width=100)
6422
6423     boton_agregar_datos = ttk.Button(ventana_Ld, text="Agregar
6424     datos",command=lambda: agregar_datos_treeview_Ld(treeview_Ld_1,
6425     combo_barras_diametro, combo_Ld, treeview_Ldc,
6426     combo_barras_ldc,combo_barras_diametro_ldc,label_Ldc_db_1,treeview_ganchos_Ldh,la
6427     bel_Ldh_traccion, label_Ldh_compresion,combo_ganchos,
6428     label_Ldh_traccion_longitud,
6429     label_Ldh_compresion_longitud,label_ganchos_check,label_icon_Ld,
6430     icon_check_on,label_anchos_columna))
6431     boton_agregar_datos.place(x=400, y=650, width=100)
6432
6433     boton_cancelar_ld = ttk.Button(ventana_Ld, text="Cancelar",
6434     command=ventana_Ld.destroy)
6435     boton_cancelar_ld.place(x=600, y=650, width=100)
6436
6437     tipo_viga_ld = label_tipo_viga_cargas.cget('text')
6438     seccion_ld = label_seccion_cargas.cget('text')
6439     luz_ld = label_seccion_luz.cget('text')
6440     wu_ld = label_seccion_Wu.cget('text')
6441
6442     entry_tipo_Ld.insert(0,tipo_viga_ld)
6443     entry_seccion_Ld.insert(0,seccion_ld)
6444     entry_luz_Ld.insert(0, f"{luz_ld} m")
6445     entry_Wu_Ld.insert(0, f"{wu_ld} kN/m")
6446
6447     labelframe_despiece = tk.LabelFrame(ventana_Ld,text="Despiece de aceros a
6448     flexión")
6449     labelframe_despiece.place(x=20,y=80,width=500,height=550)
6450     frame_despiece = tk.Frame(labelframe_despiece,bg="white")

```

```
6442     frame_despiece.pack(fill="both", expand=True)
6443
6444
6445     frame_ld= tk.Frame(ventana_Ld,bd=2,relief="ridge")
6446     frame_ld.place(x=530,y=87,width=350,height=200)
6447
6448     # Configurar un estilo personalizado para el Notebook
6449     style = ttk.Style()
6450     style.configure("CustomNotebook.TNotebook", background="lightgray",
6451     tabmargins=[0, 0, 0, 0])
6452     style.configure("CustomNotebook.TNotebook.Tab", background="white",
6453     foreground="black", padding=[10, 2])
6454     style.map("CustomNotebook.TNotebook.Tab", background=[("selected",
6455     "lightgray")])
6456
6457     # Crear el Notebook
6458     notebook_ld = ttk.Notebook(frame_ld, style="CustomNotebook.TNotebook")
6459     notebook_ld.pack(fill="both", expand=True)
6460
6461     # Crear las pestañas
6462     tab1_ld = tk.Frame(notebook_ld)
6463
6464     notebook_ld.add(tab1_ld, text="Requisitos NSR-10")
6465
6466     # --- Insertar Canvas y Scrollbar en la pestaña "Requisitos NSR-10" ---
6467     canvas_ld = tk.Canvas(tab1_ld)
6468     canvas_ld.pack(side="left", fill="both", expand=True)
6469
6470     # Crear un Scrollbar vertical para el canvas
6471     scrollbar_ld = tk.Scrollbar(tab1_ld, orient="vertical",
6472     command=canvas_ld.yview)
6473     scrollbar_ld.place(x=320,y=1,width=20,height=165)
6474
6475     # Configurar el Scrollbar para que funcione con el Canvas
6476     canvas_ld.config(yscrollcommand=scrollbar_ld.set)
6477
6478     #Crear contenido en la pestaña "Requisitos NSR-10"
6479     label_ld = tk.Label(canvas_ld,
6480     text="Nota: La NSR-10 tiene simplificado las fórmulas para
6481     el cálculo de la longitud de desarrollo de barras de acero en elementos de
6482     concreto reforzado, en función de la resistencia del concreto y del acero, el
6483     diámetro de la barra y el tipo de anclaje,según lo estipulado en el Capítulo
6484     C.12",
6485     justify="left",
6486     wraplength=300)
6487     canvas_ld.create_window(157,50, window=label_ld)
6488
6489     label_critica = tk.Label(canvas_ld,
6490     text="Sección crítica: Es la sección de la viga en la
6491     que el acero experimenta el mayor esfuerzo debido al máximo momento",
6492     justify="left",
6493     wraplength=300)
6494     canvas_ld.create_window(160,120, window=label_critica)
6495
6496     label_adherencia = tk.Label(canvas_ld,text="Condiciones para garantizar
6497     adherencia",
6498     font=("Arial", 9, "bold", "underline"))
6499     canvas_ld.create_window(160,180, window=label_adherencia)
6500
```

```

6492     label_adherencia_info = tk.Label(canvas_ld,
6493                                     text="""1. Si la barra de acero se coloca de manera
continua, se debe garantizar que la longitud mínima de desarrollo esté presente en
ambos extremos.
6494 2. La colocación de ganchos puede ser de 90° o 180°. La norma establece que se
debe garantizar una longitud mínima de desarrollo (Ldh) para los ganchos, la cual
debe ser menor que la longitud de desarrollo de la barra.
6495 3. En el caso de la colocación de dobles ganchos de lado a lado, se debe
garantizar una longitud mínima de desarrollo (Ldh) en ambos extremos de la sección
crítica. Esta longitud Ldh debe ser siempre menor que la longitud de anclaje de la
barra.""",
6496                                     justify="left",
6497                                     wraplength=300)
6498     canvas_ld.create_window(160,300, window=label_adherencia_info)
6499
6500     label_factores = tk.Label(canvas_ld,text="Factores que afectan la longitud de
desarrollo",
6501                                font=("Arial", 9, "bold", "underline"))
6502     canvas_ld.create_window(160,420, window=label_factores)
6503
6504     label_factores_info = tk.Label(canvas_ld,
6505                                    text="""1. Resistencia del concreto (f'c): A mayor
resistencia del concreto, menor será la longitud de desarrollo.
6506 2. Recubrimiento (r): Si se pone poco recubrimiento, la viga puede fallar por
adherencia.
6507 3. Separación entre barras (S): No se puede poner las barras muy juntas,por que
puede fallar por adherencia.
6508 4 Refuerzo en exceso (p): A mayor refuerzo, menor será la longitud de
desarrollo.""",
6509                                    justify="left",
6510                                    wraplength=300)
6511     canvas_ld.create_window(160,510, window=label_factores_info)
6512
6513     label_fallas = tk.Label(canvas_ld,text="Tipo de falla por adherencia",
6514                              font=("Arial", 9, "bold", "underline"))
6515     canvas_ld.create_window(160,600, window=label_fallas)
6516
6517     label_fallas_info = tk.Label(canvas_ld,
6518                                  text="1. Desprendimiento de las barras:\nHay un
desprendimientos de las barras de acero en el concreto y se presenta grietas
paralelo a la posición del acero.",
6519                                  justify="left",
6520                                  wraplength=300)
6521     canvas_ld.create_window(160,650, window=label_fallas_info)
6522
6523     label_NSR_10 = tk.Label(canvas_ld,text="La norma establece que, como mínimo,
la longitud de desarrollo debe ser igual o superior a 30 cm. No obstante, es
posible reducir esta longitud utilizando la fórmula general para calcular la
longitud de desarrollo (Ld), que se encuentra en el índice C.12.2.3., según la
siguiente fórmula:",
6524                                  justify="left",
6525                                  wraplength=300)
6526     canvas_ld.create_window(160,750, window=label_NSR_10)
6527
6528     frame_ld= tk.Frame(canvas_ld,bd=4,relief="ridge")
6529     canvas_ld.create_window(160,870, window=frame_ld,width=200,height=110)
6530
6531     # Cargar la imagen
6532     # Ruta relativa para la imagen

```

```

6533     image_ld_path = os.path.join("Iconos_proyecto", "Formula_Ld.png") # Ruta
relativa de la imagen
6534     image_ld = Image.open(image_ld_path)
6535     image_ld = image_ld.resize((200, 110)) # Redimensionar la imagen si es
necesario
6536
6537     # Convertir la imagen a un formato que Tkinter pueda manejar
6538     photo_ld = ImageTk.PhotoImage(image_ld)
6539
6540     # Insertar la imagen en un Label dentro del Frame
6541     label_ecuacion = tk.Label(frame_ld, image=photo_ld)
6542     label_ecuacion.image = photo_ld # Guardar una referencia de la imagen
6543     label_ecuacion.pack()
6544
6545     label_parametros = tk.Label(canvas_ld, text="""Parámetros de Cálculo:
6546 - f'c: Resistencia a la compresión del concreto
6547 - fy: Esfuerzo de fluencia del acero
6548 - λ: Factor por peso del concreto
6549 - ψt: Factor colocación del acero
6550 - ψe: Factor de recubrimiento epóxico
6551 - ψs: Factor del diametro del refuerzo
6552 - Cb: Distancia mínima entre barras de refuerzo longitudinal o distancia al borde
6553 - Ktr: La norma permite considerar un valor de 0 como una opción conservadora
6554 - db: Diámetro de la barra de refuerzo""",
6555                                     justify="left",
6556                                     wraplength=300)
6557     canvas_ld.create_window(160, 1030, window=label_parametros)
6558
6559     label_compresión = tk.Label(canvas_ld, text="Longitud desarrollo a
compresión", font=("Arial", 9, "bold", "underline"))
6560     canvas_ld.create_window(160, 1150, window=label_compresión)
6561
6562     label_compresión_info = tk.Label(canvas_ld, text="""La norma establece que,
para las barras sometidas a compresión, la longitud de desarrollo mínima (Ldc)
debe ser de 20 cm. Si se requiere una longitud menor, esta deberá calcularse
utilizando la fórmula:
6563 *  $Ldc = [0.24 * fy] / [\lambda * \sqrt{f'c}] * db$ 
6564 *  $Ldc = [0.043 * fy * db]$ """,
6565                                     justify="left",
6566                                     wraplength=300)
6567     canvas_ld.create_window(160, 1220, window=label_compresión_info)
6568
6569     label_ganchos_As = tk.Label(canvas_ld, text="Longitud de desarrollo para
ganchos a tracción", font=("Arial", 9, "bold", "underline"))
6570     canvas_ld.create_window(160, 1290, window=label_ganchos_As)
6571
6572     label_ganchos_As_info = tk.Label(canvas_ld, text="""La norma establece los
requisitos para los ganchos estándar sometidos a tracción en el índice C.12.5.
Existen dos tipos de ganchos: a 90° y a 180°. Para el gancho de 180°, deben
cumplirse las siguientes condiciones:
6573
6574 1- El diámetro de doblez debe cumplir con las condiciones mínimas especificadas en
la norma.
6575 2- La extensión mínima, medida desde el final del diámetro de doblez hasta el
extremo del gancho, debe ser al menos 4 veces el diámetro de la barra o, como
mínimo, 6.5 cm.\n
6576 Para el gancho de 90°, se deben cumplir las siguientes condiciones:
6577 1- Extensión mínima: Al finalizar el doblado de la barra, la extensión mínima
deberá ser 12 veces el diámetro de la barra (12db)

```

```

6578 2- Longitud de desarrollo con ganchos (Ldh): Según lo indicado en el índice
C.12.5.3 de la norma, la longitud de desarrollo con ganchos debe medirse desde el
extremo del gancho y debe ser igual o mayor a 8 veces el diámetro de la barra
(8db) o 15 cm, esta longitud debe ser suficiente para llegar hasta la sección
crítica de la barra.\n
6579 La fórmula para calcular la longitud de desarrollo con ganchos (Ldh) es: """,
6580         justify="left",
6581         wraplength=300)
6582     canvas_ld.create_window(160,1530, window=label_ganchos_As_info)
6583
6584     frame_formula_ganchos = tk.Frame(canvas_ld,bd=4,relief="ridge")
6585     canvas_ld.create_window(160,1790,
window=frame_formula_ganchos,width=200,height=40)
6586
6587     # Ruta de la imagen
6588     # Ruta relativa de la imagen
6589     ruta_imagen_gancho = os.path.join("Iconos_proyecto", "Ldh.png")
6590     # Abrir la imagen con PIL
6591     imagen_gancho = Image.open(ruta_imagen_gancho)
6592     # Redimensionar la imagen si es necesario
6593     imagen_gancho = imagen_gancho.resize((200, 40))
6594
6595     # Convertir la imagen a un formato que Tkinter pueda manejar
(ImageTk.PhotoImage)
6596     foto_gancho = ImageTk.PhotoImage(imagen_gancho)
6597
6598     # Insertar la imagen en un Label dentro del Frame
6599     label_ecuacion_gancho = tk.Label(frame_formula_ganchos, image=foto_gancho)
6600     label_ecuacion_gancho.image = foto_gancho # Mantener una referencia de la
imagen
6601     label_ecuacion_gancho.pack()
6602
6603     label_doblado = tk.Label(canvas_ld,text=""Diámetro de doblado: El diámetro de
dobrado varía según el tipo de barra y se especifica de la siguiente manera:
6604
6605 - Para barras de #3 a #8, el diámetro de doblado será igual a 6 veces el diámetro
de la barra (6db).
6606 - Para barras de #9 a #11, el diámetro de doblado será igual a 8 veces el diámetro
de la barra (8db).
6607 - Para barras de #14 a #18, el diámetro de doblado será igual a 10 veces el
diámetro de la barra (10db).""",
6608         justify="left",
6609         wraplength=300)
6610     canvas_ld.create_window(160,1900, window=label_doblado)
6611
6612     frame_calculo_Ld= tk.Frame(ventana_Ld,bd=2,relief="ridge")
6613     frame_calculo_Ld.place(x=530,y=300,width=350,height=330)
6614
6615     # Configurar un estilo personalizado para el Notebook
6616     style = ttk.Style()
6617     style.configure("CustomNotebook.TNotebook", background="lightgray",
tabmargins=[0, 0, 0, 0])
6618     style.configure("CustomNotebook.TNotebook.Tab", background="white",
foreground="black", padding=[10, 2])
6619     style.map("CustomNotebook.TNotebook.Tab", background=[("selected",
"lightgray")])
6620
6621     # Crear el Notebook
6622     notebook_calculo = ttk.Notebook(frame_calculo_Ld,
style="CustomNotebook.TNotebook")

```

```
6623     notebook_calculo.pack(fill="both", expand=True)
6624
6625     # Crear las pestañas
6626     tab1_calculo = tk.Frame(notebook_calculo)
6627
6628
6629     notebook_calculo.add(tab1_calculo, text="Cálculos previos")
6630
6631     # --- Insertar Canvas y Scrollbar en la pestaña "Requisitos NSR-10" ---
6632     canvas_calculo = tk.Canvas(tab1_calculo)
6633     canvas_calculo.pack(side="left", fill="both", expand=True)
6634
6635     # Crear un Scrollbar vertical para el canvas
6636     scrollbar_calculo = tk.Scrollbar(tab1_calculo, orient="vertical",
command=canvas_calculo.yview)
6637     scrollbar_calculo.place(x=320,y=1,width=20,height=295)
6638
6639     # Configurar el Scrollbar para que funcione con el Canvas
6640     canvas_calculo.config(yscrollcommand=scrollbar_calculo.set)
6641
6642     # Crear contenido en la pestaña "Cálculos previos"
6643     labelframe_datos = tk.LabelFrame(canvas_calculo, text="")
6644     canvas_calculo.create_window(160, 35,
window=labelframe_datos, width=300, height=50)
6645
6646     label_datos_fc = tk.Label(labelframe_datos, text="f'c:")
6647     label_datos_fc.place(x=10, y=10)
6648
6649     combo_datos_fc = ttk.Combobox(labelframe_datos, values=["17.5 Mpa", "21 Mpa",
"24.5 Mpa", "28 Mpa", "35 Mpa", "42 Mpa"])
6650     combo_datos_fc.place(x=40, y=10, width=80)
6651
6652     label_datos_fy = tk.Label(labelframe_datos, text="fy:")
6653     label_datos_fy.place(x=150, y=10)
6654
6655     combo_datos_fy = ttk.Combobox(labelframe_datos, values=["420 Mpa"])
6656     combo_datos_fy.place(x=180, y=10, width=80)
6657
6658     labelframe_calculo = tk.LabelFrame(canvas_calculo, text="")
6659     canvas_calculo.create_window(160, 170,
window=labelframe_calculo, width=300, height=200)
6660
6661     label_barras_traccion = tk.Label(labelframe_calculo, text="#Barras:")
6662     label_barras_traccion.place(x=10, y=10)
6663
6664     combo_barras_traccion = ttk.Combobox(labelframe_calculo)
6665     combo_barras_traccion.place(x=60, y=10, width=80)
6666
6667     label_diametro_barras = tk.Label(labelframe_calculo, text="ØBarras:")
6668     label_diametro_barras.place(x=10, y=160)
6669     combo_barras_diametro = ttk.Combobox(labelframe_calculo)
6670     combo_barras_diametro.place(x=60, y=160, width=80)
6671
6672     check_button_Ø = tk.Checkbutton(labelframe_calculo, text="Agregar diámetro")
6673     check_button_Ø.place(x=150, y=160)
6674
6675     check_button_Ø.config(command=agregar_diametros_barras)
6676
6677     boton_traccion = ttk.Button(labelframe_calculo, text="Agregar")
6678     boton_traccion.place(x=180, y=5, width=80)
```

```
6679
6680 # Asignar la función al botón para agregar los datos
6681 boton_traccion.config(command=lambda: agregar_datos_treeview(treeview_ld))
6682
6683 # Asignar los mismos valores de Barras_traccion a combo_barras_traccion
6684 combo_barras_traccion['values'] = Barras_traccion['values']
6685
6686 # Si hay algún valor seleccionado en Barras_traccion, seleccionarlo también en
6687 # combo_barras_traccion
6688 if Barras_traccion.get():
6689     combo_barras_traccion.set(Barras_traccion.get())
6690
6691 # Crear el Treeview para la tabla con 2 columnas
6692 treeview_ld = ttk.Treeview(labelframe_calculo, columns=("Mu", "As"),
6693 show="headings")
6694 treeview_ld.place(x=40, y=50, width=200, height=100)
6695
6696 # Definir los encabezados de las columnas
6697 treeview_ld.heading("Mu", text="Mu")
6698 treeview_ld.heading("As", text="#Barra")
6699
6700 # Insertar tres filas con valores de ejemplo para las columnas Mu y As
6701 treeview_ld.insert("", "end", values=("", ""))
6702 treeview_ld.insert("", "end", values=("", ""))
6703 treeview_ld.insert("", "end", values=("", ""))
6704
6705 def no_resize(event):
6706     # Restaurar el tamaño original de las columnas "Mu" y "As" usando un ciclo
6707     for col in treeview_ld["columns"]:
6708         treeview_ld.column(col, width=98, anchor="center")
6709
6710 # Definir el ancho de las columnas (100px como tamaño original) y evitar que
6711 # se estiren
6712 for col in treeview_ld["columns"]:
6713     treeview_ld.column(col, width=98, anchor="center", stretch=False)
6714
6715 # Vincular el evento de clic con la función no_resize
6716 treeview_ld.bind("<ButtonRelease-1>", no_resize)
6717
6718 labelframe_factores = tk.LabelFrame(canvas_calculo, text="Factores de reducción
6719 [Ld]")
6720 canvas_calculo.create_window(160, 440,
6721 window=labelframe_factores, width=300, height=320)
6722
6723 label_psi = tk.Label(labelframe_factores, text="ψt:")
6724 label_psi.place(x=60, y=10)
6725
6726 combo_psi = ttk.Combobox(labelframe_factores, values=["1.0", "1.3"])
6727 combo_psi.place(x=100, y=10, width=110)
6728 combo_psi.set("Seleccionar")
6729
6730 label_psi = tk.Label(labelframe_factores, text="ψe:")
6731 label_psi.place(x=60, y=50)
6732
6733 combo_psi = ttk.Combobox(labelframe_factores, values=["1.0"])
6734 combo_psi.place(x=100, y=50, width=110)
6735 combo_psi.set("Seleccionar")
6736
6737 label_psi = tk.Label(labelframe_factores, text="ψs:")
6738 label_psi.place(x=60, y=90)
```

```
6734
6735 combo_ψs = ttk.Combobox(labelframe_factores, values=["0.8", "1.0"])
6736 combo_ψs.place(x=100,y=90,width=110)
6737 combo_ψs.set("Seleccionar")
6738
6739 label_λ = tk.Label(labelframe_factores,text="λ:")
6740 label_λ.place(x=60,y=130)
6741
6742 combo_λ = ttk.Combobox(labelframe_factores, values=["0.75", "1.0"])
6743 combo_λ.place(x=100,y=130,width=110)
6744 combo_λ.set("Seleccionar")
6745
6746 label_Cb = tk.Label(labelframe_factores,text="Cb:")
6747 label_Cb.place(x=60,y=170)
6748
6749 combo_Cb = ttk.Combobox(labelframe_factores, values=["25.4 mm"])
6750 combo_Cb.place(x=100,y=170,width=110)
6751 combo_Cb.set("Seleccionar")
6752
6753 label_Ktr = tk.Label(labelframe_factores,text="Ktr:")
6754 label_Ktr.place(x=60,y=210)
6755
6756 combo_Ktr = ttk.Combobox(labelframe_factores, values=["0.0"])
6757 combo_Ktr.place(x=100,y=210,width=110)
6758 combo_Ktr.set("Seleccionar")
6759
6760 label_db = tk.Label(labelframe_factores,text="db:")
6761 label_db.place(x=60,y=250)
6762
6763 combo_db = ttk.Combobox(labelframe_factores, values=[])
6764 combo_db.place(x=100,y=250,width=110)
6765 combo_db.set("Seleccionar")
6766
6767 labelframe_resultados = tk.LabelFrame(canvas_calculo,text="Cálcular Ld para
barras a tracción")
6768 canvas_calculo.create_window(160,640,
window=labelframe_resultados,width=300,height=70)
6769
6770 label_Ld = tk.Label(labelframe_resultados,text="Ld:")
6771 label_Ld.place(x=30,y=10)
6772
6773 combo_Ld = ttk.Combobox(labelframe_resultados, values=[])
6774 combo_Ld.place(x=70,y=10,width=110)
6775 combo_Ld.set("Seleccionar")
6776
6777 boton_calcular_Ld = ttk.Button(labelframe_resultados,text="Calcular")
6778 boton_calcular_Ld.place(x=200,y=5,width=80)
6779
6780 boton_calcular_Ld.config(command=calcular_Ld_traccion)
6781
6782 labelframe_comprobación = tk.LabelFrame(canvas_calculo,text="Cálcular Ld para
barras a compresión Ldc ≥ 20 cm")
6783 canvas_calculo.create_window(160,790,
window=labelframe_comprobación,width=300,height=200)
6784
6785 label_Ldc = tk.Label(labelframe_comprobación,text="En el caso de barras
sometidas a compresión, la norma establece que la longitud mínima de desarrollo
(Ldc) debe ser ≥ 20 cm. Sin embargo, se deben evaluar dos condiciones para Ldc.
Dependiendo de cuál de los resultados sea más favorable, se seleccionará el que
cumpla con los requisitos para el diseño.",
```

```
6786         justify="left",
6787         wraplength=280)
6788
6789     label_Ldc.place(x=10,y=10)
6790
6791     label_barras_ldc = tk.Label(labelframe_comprobación,text="#Barras:")
6792     label_barras_ldc.place(x=10,y=140)
6793
6794     combo_barras_ldc = ttk.Combobox(labelframe_comprobación, values=[])
6795     combo_barras_ldc.place(x=60,y=140,width=80)
6796
6797     label_diametro_barras_ldc = tk.Label(labelframe_comprobación,text="ØBarras:")
6798     label_diametro_barras_ldc.place(x=150,y=140)
6799
6800     combo_barras_diametro_ldc = ttk.Combobox(labelframe_comprobación, values=[])
6801     combo_barras_diametro_ldc.place(x=206,y=140,width=80)
6802
6803     labelframe_resultados_ldc = tk.LabelFrame(canvas_calculo,text="Calcular Ldc
para barras a compresión")
6804     canvas_calculo.create_window(160,940,
window=labelframe_resultados_ldc,width=300,height=100)
6805
6806     label_fc_Ldc = tk.Label(labelframe_resultados_ldc,text="f'c:")
6807     label_fc_Ldc.place(x=10,y=10)
6808
6809     combo_fc_Ldc = ttk.Combobox(labelframe_resultados_ldc, values=["17.5 Mpa", "21
Mpa", "24.5 Mpa", "28 Mpa", "35 Mpa", "42 Mpa"])
6810     combo_fc_Ldc.place(x=40,y=10,width=80)
6811
6812     combo_fc_Ldc.bind("<<ComboboxSelected>>", Calcular_Ldc_compresion)
6813
6814     label_Ldc = tk.Label(labelframe_resultados_ldc,text="Ldc:")
6815     label_Ldc.place(x=150,y=10)
6816
6817     combo_Ldc = ttk.Combobox(labelframe_resultados_ldc, values=[])
6818     combo_Ldc.place(x=180,y=10,width=80)
6819
6820     label_Ldc_db = tk.Label(labelframe_resultados_ldc,text="")
6821     label_Ldc_db.place(x=1,y=40)
6822
6823     label_Ldc_db_1 = tk.Label(labelframe_resultados_ldc,bd=2,relief="sunken")
6824     label_Ldc_db_1.place(x=40,y=40,width=80)
6825
6826     label_18db = tk.Label(labelframe_resultados_ldc,text="18db:")
6827     label_18db.place(x=145,y=40)
6828
6829     label_18db_1 = tk.Label(labelframe_resultados_ldc,bd=2,relief="sunken")
6830     label_18db_1.place(x=180,y=40,width=80)
6831
6832     labelframe_resultados_Ldc = tk.LabelFrame(canvas_calculo,text="")
6833     canvas_calculo.create_window(160,1028,
window=labelframe_resultados_Ldc,width=300,height=60)
6834
6835     label_resultados_Ldc = tk.Label(labelframe_resultados_Ldc,text="",
justify="left",
6836     wraplength=280)
6837     label_resultados_Ldc.place(x=10,y=5)
6838
6839
6840     labelframe_ganchos = tk.LabelFrame(canvas_calculo,text="Especificaciones para
ganchos a tracción")
```

```
6841 canvas_calculo.create_window(160,1220,  
window=labelframe_ganchos,width=300,height=300)  
6842  
6843 label_ganchos_titulo = tk.Label(labelframe_ganchos,text="C.12.5.1 (Ganchos  
estandar en tracción)")  
6844 label_ganchos_titulo.place(x=10,y=10)  
6845  
6846 label_ganchos = tk.Label(labelframe_ganchos,text="Ganchos:")  
6847 label_ganchos.place(x=10,y=40)  
6848  
6849 combo_ganchos = ttk.Combobox(labelframe_ganchos, values=["90°","180°"])  
6850 combo_ganchos.place(x=70,y=40,width=100)  
6851 combo_ganchos.set("Seleccionar")  
6852  
6853 # Vincular el evento de selección al Combobox  
6854 combo_ganchos.bind("<<ComboboxSelected>>", actualizar_imagen)  
6855  
6856 frame_ganchos = tk.Frame(labelframe_ganchos,bd=4,relief="ridge",bg="white")  
6857 frame_ganchos.place(x=40,y=70,width=210,height=200)  
6858  
6859 labelframe_doblado = tk.LabelFrame(canvas_calculo,text="Especificaciones para  
diámetro de doblado")  
6860 canvas_calculo.create_window(160,1430,  
window=labelframe_doblado,width=300,height=100)  
6861  
6862 label_barra_doblado = tk.Label(labelframe_doblado,text="#Barras:")  
6863 label_barra_doblado.place(x=50,y=10)  
6864  
6865 combo_barras_doblado =  
ttk.Combobox(labelframe_doblado,values=list(barras_info_refuerzo_1.keys()))  
6866 combo_barras_doblado.place(x=120,y=10,width=100)  
6867 combo_barras_doblado.set("Seleccionar")  
6868  
6869 # Vincular el evento de selección del Combobox con la función  
diametro_de_doblado  
6870 combo_barras_doblado.bind("<<ComboboxSelected>>", diametro_de_doblado)  
6871  
6872 label_diametro_doblado = tk.Label(labelframe_doblado,text="ØDoblado:")  
6873 label_diametro_doblado.place(x=50,y=50)  
6874  
6875 label_diametro_doblado_1 = tk.Label(labelframe_doblado,bd=2,relief="sunken")  
6876 label_diametro_doblado_1.place(x=120,y=50,width=100)  
6877  
6878 labelframe_longitud_desarrollo = tk.LabelFrame(canvas_calculo,text="Longitud  
de desarrollo con ganchos")  
6879 canvas_calculo.create_window(160,1560,  
window=labelframe_longitud_desarrollo,width=300,height=140)  
6880  
6881 label_Ldh = tk.Label(labelframe_longitud_desarrollo,text="Según la NSR-10, la  
longitud de desarrollo Ldh debe ser menor que la longitud normal de  
desarrollo.Esta longitud mínima (Ldh) se mide desde el extremo del gancho hasta la  
sección crítica de la barra.La longitud mínima de Ldh debe ser al menos 8db, o 15  
cm como mínimo.",  
6882 justify="left",  
6883 wraplength=275)  
6884 label_Ldh.place(x=10,y=10)  
6885  
6886 labelframe_Calcular_Ldh = tk.LabelFrame(canvas_calculo,text="Calcular Ldh con  
ganchos")
```

```
6887 canvas_calculo.create_window(160,1710,  
window=labelframe_Calcular_Ldh,width=300,height=150)  
6888  
6889 label_concreto_Ldh = tk.Label(labelframe_Calcular_Ldh,text="Concreto:")  
6890 label_concreto_Ldh.place(x=50,y=10)  
6891  
6892 combo_concreto_Ldh = ttk.Combobox(labelframe_Calcular_Ldh, values=["Peso  
liviano", "Peso normal"])  
6893 combo_concreto_Ldh.place(x=120,y=10,width=100)  
6894  
6895 combo_concreto_Ldh.bind("<<ComboboxSelected>>", lambda event: Calcular_Ldh())  
6896  
6897 label_concreto_λ = tk.Label(labelframe_Calcular_Ldh,text="Factor [λ]:")  
6898 label_concreto_λ.place(x=50,y=40)  
6899  
6900 label_concreto_λ_factor =  
tk.Label(labelframe_Calcular_Ldh,bd=2,relief="sunken")  
6901 label_concreto_λ_factor.place(x=120,y=40,width=100)  
6902  
6903 label_factor_ψe = tk.Label(labelframe_Calcular_Ldh,text="Factor [ψe]:")  
6904 label_factor_ψe.place(x=50,y=70)  
6905  
6906 label_factor_ψe_1 = tk.Label(labelframe_Calcular_Ldh,bd=2,relief="sunken")  
6907 label_factor_ψe_1.place(x=120,y=70,width=100)  
6908  
6909 label_resultado_Ldh = tk.Label(labelframe_Calcular_Ldh,text="Ldh:")  
6910 label_resultado_Ldh.place(x=50,y=100)  
6911  
6912 label_resultado_Ldh_1 = tk.Label(labelframe_Calcular_Ldh,bd=2,relief="sunken")  
6913 label_resultado_Ldh_1.place(x=120,y=100,width=100)  
6914  
6915 labelframe_tabla_Ldh = tk.LabelFrame(canvas_calculo,text="Tabla mínimo Ldh a  
tracción")  
6916 canvas_calculo.create_window(160,1890,  
window=labelframe_tabla_Ldh,width=300,height=200)  
6917  
6918 # Crear la tabla Treeview dentro del LabelFrame  
6919 treeview_Ldh = ttk.Treeview(labelframe_tabla_Ldh, columns=("barra", "ldh",  
"traccion"), show="headings")  
6920 treeview_Ldh.place(x=50, y=10, width=200, height=80)  
6921  
6922 # Definir las columnas  
6923 treeview_Ldh.heading("barra", text="#Barra")  
6924 treeview_Ldh.heading("ldh", text="Ldh (cm)")  
6925 treeview_Ldh.heading("traccion", text="Tracción")  
6926  
6927 # Configurar el ancho de las columnas  
6928 treeview_Ldh.column("barra", width=50,anchor="center")  
6929 treeview_Ldh.column("ldh", width=50,anchor="center")  
6930 treeview_Ldh.column("traccion", width=50,anchor="center")  
6931  
6932 # Insertar una fila de ejemplo (puedes agregar más filas según necesites)  
6933 treeview_Ldh.insert("", "end", values=("", "", "As[-]"))  
6934  
6935 def no_resize(event):  
6936 # Restaurar el tamaño original de las columnas usando un ciclo  
6937 for col in treeview_Ldh["columns"]:  
6938 treeview_Ldh.column(col, width=66, anchor="center")  
6939  
6940 # Vincular el evento de clic con la función no_resize
```

```

6941     treeview_Ldh.bind("<ButtonRelease-1>", no_resize)
6942
6943     label_treeview_Ldh = tk.Label(labelframe_tabla_Ldh,text="",
6944                                 justify="left",
6945                                 wraplength=285)
6946     label_treeview_Ldh.place(x=10,y=100)
6947
6948     labelframe_tabla_Ldh_compresion = tk.LabelFrame(canvas_calculo,text="Tabla
mínimo Ldh a compresión")
6949     canvas_calculo.create_window(160,2100,
window=labelframe_tabla_Ldh_compresion,width=300,height=200)
6950
6951     # Crear la tabla Treeview dentro del LabelFrame
6952     treeview_Ldh_compresion = ttk.Treeview(labelframe_tabla_Ldh_compresion,
columns=("barra", "ldh", "compresión"), show="headings")
6953     treeview_Ldh_compresion.place(x=50, y=10, width=200, height=80)
6954
6955     # Definir las columnas
6956     treeview_Ldh_compresion.heading("barra", text="#Barra")
6957     treeview_Ldh_compresion.heading("ldh", text="Ldh (cm)")
6958     treeview_Ldh_compresion.heading("compresión", text="Compresión")
6959
6960     # Configurar el ancho de las columnas
6961     treeview_Ldh_compresion.column("barra", width=40,anchor="center")
6962     treeview_Ldh_compresion.column("ldh", width=40,anchor="center")
6963     treeview_Ldh_compresion.column("compresión", width=60,anchor="center")
6964
6965     # Insertar una fila de ejemplo (puedes agregar más filas según necesites)
6966     treeview_Ldh_compresion.insert("", "end", values=("", "", "As'))
6967
6968     # Función que previene el cambio de tamaño de las columnas
6969     def no_resize(event):
6970     # Definir los anchos específicos para cada columna
6971         treeview_Ldh_compresion.column("barra", width=50, anchor="center") #
Columna #Barra con ancho 60
6972         treeview_Ldh_compresion.column("ldh", width=60, anchor="center")      #
Columna Ldh con ancho 60
6973         treeview_Ldh_compresion.column("compresión", width=87, anchor="center") #
Columna Compresión con ancho 70
6974
6975     # Vincular el evento de clic con la función no_resize
6976     treeview_Ldh_compresion.bind("<ButtonRelease-1>", no_resize)
6977
6978     label_treeview_Ldh_compresion =
tk.Label(labelframe_tabla_Ldh_compresion,text="",justify="left",wraplength=285)
6979     label_treeview_Ldh_compresion.place(x=10,y=100)
6980
6981     labelframe_ganchos_estandar = tk.LabelFrame(canvas_calculo,text="Calcular
Longitud del gancho mínimo")
6982     canvas_calculo.create_window(160,2320,
window=labelframe_ganchos_estandar,width=300,height=220)
6983
6984     # Crear la tabla Treeview dentro del LabelFrame
6985     treeview_ganchos = ttk.Treeview(labelframe_ganchos_estandar, columns=("barra",
"gancho", "longitud"), show="headings")
6986     treeview_ganchos.place(x=50, y=10, width=200, height=80)
6987
6988     # Definir las columnas
6989     treeview_ganchos.heading("barra", text="#Barra")
6990     treeview_ganchos.heading("gancho", text="Gancho")

```

```

6991     treeview_ganchos.heading("longitud", text="Longitud")
6992
6993     # Configurar el ancho de las columnas
6994     treeview_ganchos.column("barra", width=50, anchor="center")
6995     treeview_ganchos.column("gancho", width=50, anchor="center")
6996     treeview_ganchos.column("longitud", width=50, anchor="center")
6997
6998     # Insertar una fila de ejemplo (puedes agregar más filas según necesites)
6999     treeview_ganchos.insert("", "end", values=("", " ", ""))
7000     treeview_ganchos.insert("", "end", values=("", " ", ""))
7001
7002     label_treeview_ganchos =
tk.Label(labelframe_ganchos_estandar, text="", justify="left", wraplength=280)
7003     label_treeview_ganchos.place(x=10, y=100)
7004
7005     def no_resize(event):
7006         # Restaurar el tamaño original de las columnas usando un ciclo
7007         for col in treeview_ganchos["columns"]:
7008             treeview_ganchos.column(col, width=66, anchor="center")
7009
7010         # Vincular el evento de clic con la función no_resize
7011         treeview_ganchos.bind("<ButtonRelease-1>", no_resize)
7012
7013         # Función para ajustar el tamaño del canvas y el área de desplazamiento
7014         def update_scrollregion_calculo(event=None):
7015             bbox_calculo = canvas_calculo.bbox("all") # Obtiene las coordenadas del
área de todo el contenido
7016             if bbox_calculo:
7017                 # Añadir un margen adicional de 100 píxeles a las coordenadas del área
7018                 canvas_calculo.config(scrollregion=(0, 0, bbox_calculo[2] + 100,
bbox_calculo[3] + 10))
7019
7020             # Llamar a update_scrollregion para ajustar el área de desplazamiento
7021             canvas_calculo.bind("<Configure>", update_scrollregion_calculo)
7022
7023             # Actualizar el área de scroll después de agregar contenido
7024             canvas_calculo.update_idletasks()
7025             canvas_calculo.config(scrollregion=canvas_calculo.bbox("all"))
7026
7027             # Función para ajustar el tamaño del canvas y el área de desplazamiento
7028             def update_scrollregion(event=None):
7029                 bbox_ld = canvas_ld.bbox("all") # Obtiene las coordenadas del área de
todo el contenido
7030                 if bbox_ld:
7031                     # Añadir un margen adicional de 100 píxeles a las coordenadas del área
7032                     canvas_ld.config(scrollregion=(0, 0, bbox_ld[2] + 100, bbox_ld[3] +
10))
7033
7034                 # Llamar a update_scrollregion para ajustar el área de desplazamiento
7035                 canvas_ld.bind("<Configure>", update_scrollregion)
7036
7037                 # Actualizar el área de scroll después de agregar contenido
7038                 canvas_ld.update_idletasks()
7039                 canvas_ld.config(scrollregion=canvas_ld.bbox("all"))
7040
7041
7042     def agregar_datos_treeview(treeview_ld):
7043         # Obtener todos los valores seleccionados en el ComboBox Momento_Mu
7044         momento_mu_values = Momento_Mu['values'] # Momento_Mu es el ComboBox de la
ventana principal

```

```

7045
7046     # Obtener los valores seleccionados en el ComboBox combo_barras_traccion
7047     barras_traccion_values = combo_barras_traccion['values'] #
combo_barras_traccion es otro ComboBox
7048
7049     # Verificar si ambos ComboBoxes tienen al menos tres valores
7050     if len(momento_mu_values) >= 3 and len(barras_traccion_values) >= 3:
7051         # Asignar los valores a las filas del Treeview
7052         for i in range(3): # Recorremos las 3 filas
7053             # Actualizar la primera columna "Mu" con los valores de Momento_Mu
7054             # Actualizar la segunda columna "As" con los valores de
combo_barras_traccion
7055             treeview_ld.item(treeview_ld.get_children()[i], values=
(momento_mu_values[i], barras_traccion_values[i]))
7056
7057         # Ahora actualizamos las etiquetas con los valores de la columna "As"
7058         # Asignar el valor de la primera fila en la columna "As" a
label_As_negativo_izquierda
7059         label_As_negativo_izquierda.config(text=barras_traccion_values[0])
7060
7061         # Asignar el valor de la segunda fila en la columna "As" a
label_As_positivo_centro
7062         label_As_positivo_centro.config(text=barras_traccion_values[1])
7063
7064         # Asignar el valor de la tercera fila en la columna "As" a
label_As_negativo_derecha
7065         label_As_negativo_derecha.config(text=barras_traccion_values[2])
7066
7067 def agregar_diametros_barras():
7068     # Obtener todos los valores del combo_barras_traccion (sin separar por coma)
7069     valores_traccion = combo_barras_traccion['values'] # Obtiene la lista
completa de valores
7070
7071     # Lista para almacenar los diámetros correspondientes
7072     diametros = []
7073
7074     # Recorrer cada valor de barra de tracción seleccionada
7075     for valor in valores_traccion:
7076         # Extraer el número de la barra (último número después del "#")
7077         if "#" in valor:
7078             numero_barra = valor.split("#")[-1]
7079
7080         # Buscar el diámetro de esa barra en el diccionario
7081         for barra, info in barras_info_refuerzo.items():
7082             if f"No.{numero_barra}" in barra:
7083                 diametros.append(f"Ø {info['diametro_mm']} mm")
7084             break
7085
7086     # Actualizar los valores del combo_barras_diametro con los diámetros
encontrados
7087     combo_barras_diametro['values'] = diametros # Asigna la lista completa de
diámetros a combo_barras_diametro
7088
7089     # Si hay diámetros disponibles, se selecciona el primero de ellos por defecto
(opcional)
7090     if diametros:
7091         combo_barras_diametro.set(diametros[0]) # Selecciona el primer valor por
defecto
7092
7093     # Actualizar los valores del combo_db con los mismos diámetros encontrados

```

```
7094     combo_db['values'] = diametros # Asigna la lista completa de diámetros a
      combo_db
7095
7096     # Si hay diámetros disponibles, se selecciona el primero de ellos por defecto
      (opcional)
7097     if diametros:
7098         combo_db.set(diametros[0]) # Selecciona el primer valor por defecto
7099
7100     # ---- Lógica para actualizar combo_barras_ldc ----
7101
7102     # Obtener el valor seleccionado de Barras_compresion
7103     valor_compresion = Barras_compresion.get()
7104
7105     # Lista para almacenar los valores correspondientes en combo_barras_ldc
7106     valores_ldc = []
7107
7108     if valor_compresion:
7109         # Solo añadimos el valor seleccionado en Barras_compresion a
      combo_barras_ldc
7110         valores_ldc.append(valor_compresion)
7111
7112     # Actualizar los valores del combo_barras_ldc con los valores encontrados
7113     combo_barras_ldc['values'] = valores_ldc
7114
7115     # Si hay valores disponibles, seleccionar el primero (opcional)
7116     if valores_ldc:
7117         combo_barras_ldc.set(valores_ldc[0]) # Selecciona el primer valor por
      defecto
7118
7119     # ---- Nueva lógica para actualizar combo_barras diametro_ldc ----
7120
7121     # Obtener el valor seleccionado de combo_barras_ldc (que es el mismo que el de
      Barras_compresion)
7122     valor_ldc = combo_barras_ldc.get()
7123
7124     # Lista para almacenar los diámetros correspondientes en
      combo_barras_diametro_ldc
7125     diametros_ldc = []
7126
7127     if valor_ldc:
7128         # Extraemos el último dígito (el número después del "#") del valor
      seleccionado en combo_barras_ldc
7129         numero_barra_ldc = valor_ldc.split("#")[-1]
7130
7131         # Buscar el diámetro correspondiente en el diccionario
      barras_info_refuerzo
7132         for barra, info in barras_info_refuerzo.items():
7133             if f"No.{numero_barra_ldc}" in barra:
7134                 # Añadir el diámetro correspondiente a la lista diametros_ldc
7135                 diametros_ldc.append(f"Ø {info['diametro_mm']} mm")
7136                 break
7137
7138         # Actualizar el combo_barras_diametro_ldc con el diámetro encontrado
7139         combo_barras_diametro_ldc['values'] = diametros_ldc
7140
7141         # Si hay un diámetro disponible, seleccionamos el primero (opcional)
7142         if diametros_ldc:
7143             combo_barras_diametro_ldc.set(diametros_ldc[0]) # Selecciona el primer
      valor por defecto
7144
```

```

7145
7146 def calcular_Ld_traccion():
7147     # Obtener los valores de los combos
7148     fy = 420
7149     fc = combo_datos_fc.get() # Obtener el valor de fc
7150     fc = float(fc.split()[0]) # Extraer solo el valor numérico (ej. "17.5")
7151
7152     # Obtener los valores de los factores
7153     ψt = float(combo_ψt.get()) # ψt
7154     ψe = float(combo_ψe.get()) # ψe
7155     ψs = float(combo_ψs.get()) # ψs
7156     λ = float(combo_λ.get())   # λ
7157
7158     # Extraer el valor numérico de Cb sin la unidad "mm"
7159     Cb_str = combo_Cb.get() # Valor con unidad (ej. "25.4 mm")
7160     Cb = float(Cb_str.split()[0]) # Obtener solo la parte numérica antes del
espacio
7161
7162     db_values = combo_db['values'] # Obtener los diámetros de combo_db
7163
7164     # Lista para almacenar los resultados redondeados
7165     resultados_Ld = []
7166
7167     # Calcular Ld_tracción para cada valor de db
7168     for db_value in db_values:
7169         db_value_numeric = float(db_value.split()[1]) # Extraer solo el valor
numérico (ej. "15.9")
7170
7171         # Calcular Ld_tracción
7172         Ld_traccion = ((fy / (1.1 * λ * (fc**0.5))) * (ψt * ψe * ψs) / Cb) *
db_value_numeric
7173
7174         # Redondear al siguiente entero mayor
7175         Ld_traccion_redondeado = math.ceil(Ld_traccion)
7176
7177         # Convertir a "cm" y agregar a la lista de resultados
7178         resultados_Ld.append(f"{Ld_traccion_redondeado} cm")
7179
7180     # Actualizar el combo_Ld con los resultados calculados
7181     combo_Ld['values'] = resultados_Ld # Asignar la lista de resultados con "cm"
7182
7183     # Si hay resultados disponibles, seleccionar el primero (opcional)
7184     if resultados_Ld:
7185         combo_Ld.set(resultados_Ld[0]) # Selecciona el primer valor por defecto
7186
7187     # Actualizar el valor en el label de diámetro de tracción
7188     db_seleccionado = combo_db.get()
7189     label_diametro_traccion.config(text=f"{db_seleccionado}")
7190
7191     # Actualizar el valor en el label de Ld tracción
7192     Ld_seleccionado = combo_Ld.get()
7193     label_Ld_traccion.config(text=f"{Ld_seleccionado}")
7194
7195 def Calcular_Ldc_compresion(event):
7196     # Obtener el valor de f'c seleccionado
7197     f_c_Ld = combo_fc_Ldc.get()
7198
7199     # Obtener el valor de db_Ld desde combo_barras_diametro_ldc
7200     db_Ld = combo_barras_diametro_ldc.get()
7201

```

```

7202 # Convertir db_Ld de texto a número (extraer solo la parte numérica)
7203 if db_Ld:
7204     try:
7205         # Extraer el número (15.9 de "Ø 15.9 mm")
7206         db_Ld_value = float(db_Ld.split()[1])
7207         db_Ld_value = db_Ld_value / 10 # Dividir el valor entre 10
7208     except ValueError:
7209         db_Ld_value = 0 # Si no se puede convertir, asignar 0
7210 else:
7211     db_Ld_value = 0
7212
7213 # Convertir f'c de texto a número (Mpa) - extraer el número
7214 if f_c_Ld:
7215     try:
7216         f_c_value = float(f_c_Ld.split()[0]) # Extraer el número de "Mpa"
7217
7218         # Calcular Ldc
7219         Ldc_compresion = 0.24 * 420 / math.sqrt(f_c_value)
7220
7221         # Redondear Ldc al entero más cercano
7222         Ldc_compresion_redondeado = round(Ldc_compresion)
7223
7224         # Formatear el resultado como "{resultado}db"
7225         resultado = f"{Ldc_compresion_redondeado} db"
7226
7227         # Actualizar el combo_Ldc con el resultado
7228         combo_Ldc.set(resultado)
7229
7230         # Actualizar el label_Ldc_db con el mismo resultado pero con el
formato "{resultado}:"
7231         label_Ldc_db.config(text=f"{resultado}:")
7232
7233         # Realizar el cálculo Ldc_compresion * db_Ld
7234         resultado_Ldc_db_Ld = Ldc_compresion_redondeado * db_Ld_value
7235
7236         # Redondear el resultado de Ldc * db_Ld al número entero más cercano
7237         resultado_Ldc_db_Ld_redondeado = round(resultado_Ldc_db_Ld)
7238
7239         # Actualizar el label_Ldc_db_1 con el nuevo resultado (entero y con
"cm")
7240         label_Ldc_db_1.config(text=f"{resultado_Ldc_db_Ld_redondeado} cm")
7241
7242         # Calcular 18 * db_Ld_value
7243         resultado_18db = 18 * db_Ld_value
7244
7245         # Redondear el resultado de 18db
7246         resultado_18db_redondeado = round(resultado_18db)
7247
7248         # Actualizar el label_18db_1 con el resultado
7249         label_18db_1.config(text=f"{resultado_18db_redondeado} cm")
7250
7251         # Comparar los dos valores para ver cuál es mayor
7252         if resultado_Ldc_db_Ld_redondeado > resultado_18db_redondeado:
7253             mayor_resultado = resultado_Ldc_db_Ld_redondeado
7254         else:
7255             mayor_resultado = resultado_18db_redondeado
7256
7257         # Mostrar el resultado mayor en el label de resultados
7258         label_resultados_Ldc.config(

```

```

7259         text=f"Con un concreto de {f_c_Ld} se requiere una longitud mínima
de desarrollo a compresión de {mayor_resultado} cm"
7260     )
7261
7262     # ACTUALIZAR EL LABEL DE DIÁMETRO DE COMPRESIÓN
7263     label_diametro_compresion.config(text=f"{db_Ld}")
7264
7265     # ACTUALIZAR EL LABEL DE LDC COMPRESIÓN
7266     label_Ldc_compresion.config(text=f"{mayor_resultado} cm")
7267
7268     except ValueError:
7269         pass # No hacemos nada en caso de error
7270
7271 # Función que se ejecuta al seleccionar una opción en el Combobox
7272 def actualizar_imagen(event):
7273     seleccion = combo_ganchos.get() # Obtenemos la opción seleccionada
7274
7275     # Eliminar la imagen previa si existe
7276     for widget in frame_ganchos.winfo_children():
7277         widget.destroy()
7278
7279     # Definir la ruta de la imagen según la selección
7280     if seleccion == "90°":
7281         ruta_imagen = os.path.join("Iconos_proyecto", "gancho_90.png")
7282     elif seleccion == "180°":
7283         ruta_imagen = os.path.join("Iconos_proyecto", "gancho_180.png")
7284     else:
7285         return # Si no hay selección válida, no hacemos nada
7286
7287     # Cargar la imagen
7288     imagen = Image.open(ruta_imagen)
7289     imagen = imagen.resize((200, 200))
7290
7291     # Convertir la imagen a un formato que pueda mostrar Tkinter
7292     imagen_tk = ImageTk.PhotoImage(imagen)
7293
7294     # Mostrar la imagen dentro del frame
7295     etiqueta_imagen = tk.Label(frame_ganchos, image=imagen_tk)
7296     etiqueta_imagen.image = imagen_tk # Necesario para evitar que la imagen se
elimine automáticamente
7297     etiqueta_imagen.pack()
7298
7299 # Función para calcular el diámetro de doblado
7300 def diametro_de_doblado(event):
7301     seleccion_doblado = combo_barras_doblado.get() # Obtenemos la barra
seleccionada
7302     barras_info_doblado = barras_info_refuerzo_1.get(seleccion_doblado) #
Obtenemos la información de la barra seleccionada
7303
7304     if barras_info_doblado:
7305         db_doblado = barras_info_doblado['diametro_mm'] # Extraemos el valor del
diámetro en mm
7306
7307         if seleccion_doblado in barras_info_refuerzo_1:
7308             if seleccion_doblado in ["Barra No.2", "Barra No.3", "Barra No.4",
"Barra No.5", "Barra No.6", "Barra No.7", "Barra No.8"]:
7309                 # Fórmula para barras No.2 a No.8: (6 * db_doblado) / 10
7310                 resultado_doblado = (6 * db_doblado) / 10
7311             elif seleccion_doblado in ["Barra No.9", "Barra No.10", "Barra
No.11"]:

```

```
7312         # Fórmula para barras No.9 a No.11: (8 * db_doblado) / 10
7313         resultado_doblado = (8 * db_doblado) / 10
7314         elif seleccion_doblado in ["Barra No.14", "Barra No.18"]:
7315             # Fórmula para barras No.14 a No.18: (10 * db_doblado) / 10
7316             resultado_doblado = (10 * db_doblado) / 10
7317
7318         # Mostrar el resultado sin decimales
7319         label_diametro_doblado_1.config(text=f"{int(resultado_doblado)} cm")
7320
7321 def Calcular_Ldh():
7322     # Obtener valores de los combos
7323     tipo_concreto = combo_concreto_Ldh.get()
7324     fc_Ldh = combo_fc_Ldc.get()
7325     db_Ldh = combo_barras_diametro.get()
7326     db_Ldh_traccion = combo_barras_traccion.get()
7327     barra_compresion = Barras_compresion.get()
7328     gancho_seleccionado = combo_ganchos.get()
7329
7330     # Verificar si se han seleccionado todos los campos necesarios
7331     if not tipo_concreto or not fc_Ldh or not db_Ldh or not barra_compresion or
not gancho_seleccionado:
7332         label_resultado_Ldh_1.config(text="Faltan datos")
7333         return
7334
7335     # Establecer constantes
7336     psi_e_Ldh = 1.0 # valor constante
7337     fy_Ldh = 420 # valor constante
7338
7339     # Asignar λ_Ldh basado en la opción de concreto
7340     if tipo_concreto == "Peso liviano":
7341         lambda_Ldh = 0.75
7342         label_concreto_lambda_factor.config(text=f"{lambda_Ldh}")
7343     elif tipo_concreto == "Peso normal":
7344         lambda_Ldh = 1.0
7345         label_concreto_lambda_factor.config(text=f"{lambda_Ldh}")
7346     else:
7347         label_resultado_Ldh_1.config(text="Opción de concreto no válida")
7348         return
7349
7350     # Actualizar el valor de ψe (constantemente 1.0 en este caso)
7351     label_factor_pse_1.config(text=f"{psi_e_Ldh}")
7352
7353     # Obtener el valor de f'c_Ldh a partir del combo_fc_Ldc
7354     if fc_Ldh == "17.5 Mpa":
7355         fc_Ldh_value = 17.5
7356     elif fc_Ldh == "21 Mpa":
7357         fc_Ldh_value = 21
7358     elif fc_Ldh == "24.5 Mpa":
7359         fc_Ldh_value = 24.5
7360     elif fc_Ldh == "28 Mpa":
7361         fc_Ldh_value = 28
7362     elif fc_Ldh == "35 Mpa":
7363         fc_Ldh_value = 35
7364     elif fc_Ldh == "42 Mpa":
7365         fc_Ldh_value = 42
7366     else:
7367         label_resultado_Ldh_1.config(text="Opción de resistencia no válida")
7368         return
7369
7370     # Extraer el valor numérico del diámetro de barra seleccionada
```

```

7371     # Usamos una expresión regular para capturar solo el número en el formato "Ø X
mm"
7372     match = re.search(r'(\d+(\.\d+)?)', db_Ldh)
7373     if match:
7374         db_Ldh_value = float(match.group(1)) # Convertir a float el valor
numérico
7375     else:
7376         label_resultado_Ldh_1.config(text="Diámetro de barra no válido")
7377         return
7378
7379     # Calcular Ldh_desarrollo utilizando la fórmula
7380     Ldh_desarrollo = (0.24 * psi_e_Ldh * fy_Ldh) / (lambda_Ldh *
math.sqrt(fc_Ldh_value)) * db_Ldh_value
7381
7382     # Dividir el resultado por 10
7383     Ldh_desarrollo /= 10
7384
7385     # Convertir el resultado a entero (sin decimales)
7386     Ldh_desarrollo_entero = int(Ldh_desarrollo)
7387
7388     # Redondear el resultado si la opción es "Peso normal" (al múltiplo de 5 más
cercano)
7389     if tipo_concreto == "Peso normal":
7390         # Redondear al múltiplo de 5 más cercano
7391         Ldh_desarrollo_entero = round(Ldh_desarrollo_entero / 5) * 5
7392
7393     # Mostrar el resultado en el label sin decimales
7394     label_resultado_Ldh_1.config(text=f"{Ldh_desarrollo_entero} cm")
7395
7396     # Ahora actualizas el label_Ldh_traccion con el mismo valor o resultado
7397     label_Ldh_traccion.config(text=f"{Ldh_desarrollo_entero} cm")
7398
7399     # Actualizar el label_ldh_t con el valor de Ldh_desarrollo_entero
7400     label_ldh_t.config(text=f"{Ldh_desarrollo_entero} cm")
7401
7402     # Ahora actualizar el Treeview con los datos
7403     treeview_Ldh.item(treeview_Ldh.get_children()[0], values=(db_Ldh_traccion,
Ldh_desarrollo_entero, "As[-]"))
7404
7405     # Actualizar el label con el formato requerido
7406     label_treeview_Ldh.config(
7407         text=(
7408             f"Para el diseño de la viga con concreto de {fc_Ldh} y acero
{db_Ldh_traccion} "
7409             f"se requiere una longitud mínima Ldh de {Ldh_desarrollo_entero} cm
para garantizar "
7410             f"el anclaje adecuado del acero en el concreto y un aporte de
adherencia necesario para el buen desempeño estructural."
7411         )
7412     )
7413
7414     # Obtener el número de la barra de compresión desde el combo
7415     barra_numero = barra_compresion.split('#')[-1] # Obtener el último número
7416
7417     # Buscar el valor del diámetro de la barra en el diccionario
7418     barra_key = f"Barra No.{barra_numero}"
7419     if barra_key in barras_info_refuerzo_1:
7420         barra_diametro = barras_info_refuerzo_1[barra_key]["diametro_mm"]
7421

```

```
7422         # Calcular Ldh para compresión usando la misma fórmula que para la
tracción
7423         Ldh_desarrollo_compresion = (0.24 * psi_e_Ldh * fy_Ldh) / (lambda_Ldh *
math.sqrt(fc_Ldh_value)) * barra_diametro
7424         Ldh_desarrollo_compresion /= 10 # Dividir por 10 para convertir en cm
7425
7426         # Convertir el resultado de compresión a entero (sin decimales)
7427         Ldh_desarrollo_compresion_entero = int(Ldh_desarrollo_compresion)
7428
7429         # Redondear al múltiplo de 5 más cercano si es "Peso normal"
7430         if tipo_concreto == "Peso normal":
7431             Ldh_desarrollo_compresion_entero =
round(Ldh_desarrollo_compresion_entero / 5) * 5
7432
7433         label_Ldh_compresion.config(text=f"{Ldh_desarrollo_compresion_entero} cm")
7434
7435         # Actualizar el label_ldh_c con el valor de
Ldh_desarrollo_compresion_entero
7436         label_ldh_c.config(text=f"{Ldh_desarrollo_compresion_entero} cm")
7437
7438         # Actualizar la segunda columna del treeview_Ldh_compresion con el cálculo
de Ldh para compresión
7439         treeview_Ldh_compresion.item(treeview_Ldh_compresion.get_children()[0],
values=(barra_compresion, Ldh_desarrollo_compresion_entero, "As"))
7440
7441         # Mostrar el resultado con gancho
7442         label_treeview_Ldh_compresion.config(
7443             text=(f"Para la barra de compresión {barra_compresion}, se requiere
una longitud mínima Ldh de "
7444                 f"{Ldh_desarrollo_compresion_entero} cm para el anclaje adecuado
en el concreto. "
7445                 f"Además, se procederá con las configuraciones del gancho
seleccionado de {gancho_seleccionado}.")
7446         )
7447
7448         # Calcular gancho para tracción
7449         if gancho_seleccionado == "90°":
7450             gancho_traccion = 12 * db_Ldh_value / 10
7451         elif gancho_seleccionado == "180°":
7452             gancho_traccion = 4 * db_Ldh_value / 10
7453         else:
7454             gancho_traccion = 0
7455
7456         # Redondear el gancho para tracción al múltiplo de 5
7457         gancho_traccion_entero = round(gancho_traccion / 5) * 5
7458
7459         # Calcular gancho para compresión
7460         if gancho_seleccionado == "90°":
7461             gancho_compresion = 12 * barra_diametro / 10
7462         elif gancho_seleccionado == "180°":
7463             gancho_compresion = 4 * barra_diametro / 10
7464         else:
7465             gancho_compresion = 0
7466
7467         # Redondear el gancho para compresión al múltiplo de 5
7468         gancho_compresion_entero = round(gancho_compresion / 5) * 5
7469
7470         # Actualizar label_Ldh_traccion_longitud con el valor de gancho_traccion
7471         label_Ldh_traccion_longitud.config(text=f"{gancho_traccion_entero} cm")
7472
```

```
7473     # Actualizar label_Ldh_compresion_longitud con el valor de
gancho_compresion
7474     label_Ldh_compresion_longitud.config(text=f"{gancho_compresion_entero}
cm")
7475
7476     # Actualizar Treeview para ganchos con valores redondeados
7477     treeview_ganchos.item(treeview_ganchos.get_children()[0], values=
(db_Ldh_traccion, gancho_seleccionado, f"{gancho_traccion_entero} cm"))
7478     treeview_ganchos.item(treeview_ganchos.get_children()[1], values=
(barra_compresion, gancho_seleccionado, f"{gancho_compresion_entero} cm"))
7479
7480     # Actualizar el label con el mensaje adicional
7481     label_treeview_ganchos.config(
7482         text=(
7483             f"En sistemas de pórticos resistentes diseñados para soportar
cargas gravitacionales o sismo, "
7484             f"se recomienda garantizar un anclaje adecuado tanto para el acero
superior como para el acero inferior. "
7485             f"Los ganchos calculados son: Tracción = {gancho_traccion_entero}
cm, Compresión = {gancho_compresion_entero} cm."
7486         )
7487     )
7488
7489     # Actualizar label_gancho_anclaje con el valor del gancho seleccionado
7490     label_gancho_anclaje.config(text=f"{gancho_seleccionado}")
7491
7492     # Actualizar label_longitud_t con el valor de gancho_traccion_entero
7493     label_longitud_t.config(text=f"{gancho_traccion_entero} cm")
7494
7495     # Actualizar label_longitud_c con el valor de gancho_compresion_entero
7496     label_longitud_c.config(text=f"{gancho_compresion_entero} cm")
7497
7498
7499 def agregar_datos_treeview_Ld(treeview_Ld_1, combo_barras_diametro, combo_Ld,
treeview_Ldc, combo_barras_ldc, combo_barras_diametro_ldc, label_Ldc_db_1,
7500     treeview_ganchos_Ldh, label_Ldh_traccion,
label_Ldh_compresion, combo_ganchos, label_Ldh_traccion_longitud, label_Ldh_compresio
n_longitud,
7501     label_ganchos_check, label_icon_Ld,
icon_check_on, label_ancho_columna):
7502     # Obtener todos los valores del ComboBox
7503     barras_traccion_values = Barras_traccion['values'] # Lista de valores del
ComboBox
7504
7505     # Obtener todos los valores del ComboBox combo_barras_diametro
7506     barras_diametro_values = combo_barras_diametro['values'] # Lista de valores
del ComboBox combo_barras_diametro
7507
7508     # Obtener todos los valores del ComboBox combo_Ld
7509     Ld_values = combo_Ld['values'] # Lista de valores del ComboBox combo_Ld
7510
7511     # Obtener todos los valores del ComboBox combo_barras_ldc
7512     barras_ldc_values = combo_barras_ldc['values'] # ComboBox combo_barras_ldc
7513
7514     # Obtener el valor del combo_barras_diametro_ldc
7515     barra_diametro_ldc_value = combo_barras_diametro_ldc.get()
7516
7517     # Obtener el texto del Label label_Ldc_db_1
7518     label_text = label_Ldc_db_1.cget("text") # Obtener el texto del label
7519
```

```
7520     # Obtener el texto de los labels de tracción y compresión
7521     label_traccion_text = label_Ldh_traccion.cget("text") # Obtener el texto de
label_Ldh_traccion
7522     label_compresion_text = label_Ldh_compresion.cget("text") # Obtener el texto
de label_Ldh_compresion
7523
7524     # Obtener el valor de combo_ganchos (para las columnas 4 de las filas 1 y 2)
7525     valor_combo_ganchos = combo_ganchos.get() # Valor seleccionado en el
combo_ganchos
7526
7527     # Obtener los textos de longitud para tracción y compresión
7528     traccion_longitud_text = label_Ldh_traccion_longitud.cget("text") # Longitud
de tracción
7529     compresion_longitud_text = label_Ldh_compresion_longitud.cget("text") #
Longitud de compresión
7530
7531     # Verificar si todos los ComboBoxes tienen al menos 3 valores (excepto
combo_barras_ldc)
7532     if len(barras_traccion_values) >= 3 and len(barras_diametro_values) >= 3 and
len(Ld_values) >= 3:
7533         # Verificar si el combo_barras_ldc tiene solo un valor
7534         if len(barras_ldc_values) == 1:
7535             barra_ldc = barras_ldc_values[0] # Usar el único valor disponible en
combo_barras_ldc
7536
7537         # Actualizar las filas de ambos Treeviews
7538         for i in range(3): # Solo recorreremos 3 filas para cada Treeview
7539             barra_traccion = barras_traccion_values[i]
7540             barra_diametro = barras_diametro_values[i]
7541             Ld_value = Ld_values[i]
7542
7543             # Actualizar el primer Treeview (treeview_Ld_1)
7544             if i == 0:
7545                 treeview_Ld_1.item(fila1_id, values=("As[-]", barra_traccion,
barra_diametro, Ld_value))
7546             elif i == 1:
7547                 treeview_Ld_1.item(fila2_id, values=("As[+]", barra_traccion,
barra_diametro, Ld_value))
7548             elif i == 2:
7549                 treeview_Ld_1.item(fila3_id, values=("As[-]", barra_traccion,
barra_diametro, Ld_value))
7550
7551             # Actualizar el segundo Treeview (treeview_Ldc) - solo la primera
fila
7552             treeview_Ldc.item(fila1_ldc, values=("As'", barra_ldc,
barra_diametro_ldc_value, label_text))
7553
7554             # Actualizar el tercer Treeview (treeview_ganchos_Ldh)
7555             # Primera fila: insertar el primer valor de Barras_traccion en la
segunda columna
7556             treeview_ganchos_Ldh.item(fila_gancho_1, values=("Tracción",
barras_traccion_values[0], label_traccion_text, valor_combo_ganchos,
traccion_longitud_text))
7557
7558             # Segunda fila: insertar el primer valor de combo_barras_ldc en la
segunda columna
7559             treeview_ganchos_Ldh.item(fila_gancho_2, values=("Compresión",
barras_ldc_values[0], label_compresion_text, valor_combo_ganchos,
compresion_longitud_text))
7560
```

```

7561         # Cálculo del ancho mínimo de la columna
7562         recubrimiento_columna = 5 # Parámetro constante
7563         try:
7564             # Usamos una expresión regular para extraer el valor numérico de
7565             'label_traccion_text'
7566             match = re.match(r"(\d+(\.\d+)?)\s*cm", label_traccion_text)
7567             # Busca números seguidos de " cm"
7568             if match:
7569                 label_traccion_value = float(match.group(1)) # Convertir
7570                 la parte numérica a float
7571                 # Realizamos el cálculo de Ancho_Columna
7572                 Ancho_Columna = label_traccion_value +
7573                 recubrimiento_columna
7574             else:
7575                 # Si no se encuentra un número, usamos un valor predeterminado
7576                 Ancho_Columna = recubrimiento_columna
7577             except ValueError:
7578                 # Si hay algún error con la conversión, solo usamos
7579                 recubrimiento_columna
7580                 Ancho_Columna = recubrimiento_columna
7581
7582             # Actualizar el label_ganchos_check con el cálculo del ancho de la
7583             columna
7584             label_ganchos_check.config(text=f"La dimensión requerida para el
7585             ancho mínimo de la columna es de {Ancho_Columna} cm. Hasta este momento hemos
7586             completado el diseño de flexión. Ahora puedes proceder con las configuraciones
7587             correspondientes al diseño por cortante.")
7588
7589             # Actualizar la imagen del icono
7590             label_icon_Ld.config(image=icon_check_on)
7591
7592             # Actualizar el valor de Ancho_Columna en el label_ancho_columna
7593             label_ancho_columna.config(text=f"{Ancho_Columna} cm")
7594
7595             # Diseño Cortante - Funciones de cálculo
7596             # Variable para controlar si la ventana secundaria está abierta
7597             lineas_discontinuas = []
7598             posiciones_x = []
7599             cotas = []
7600             textos_cotas = []
7601             # Variable para guardar el valor anterior de entry_borde_C
7602             ultimo_valor_borde_C = None
7603
7604             # Variable global para verificar si la subventana_refuerzo ya fue abierta
7605             subventana_abierta_cortante = False
7606
7607             # Diccionario para llevar el control del último estribo dibujado en cada grilla
7608             ultimo_estribo = {"derecha": None, "izquierda": None}
7609             estribos_plots = []
7610             cotas_plots = []
7611             etiquetas_plots = []
7612             # Función para abrir la subventana
7613             def abrir_subventana_refuerzo():
7614                 global
7615                 subventana_abierta_cortante, combobox_estribo_st, entry_diametro_sr, treeview_control
7616                 , menu_opciones, entry_UN_flejes, entry_UN_luz
7617                 global
7618                 entry_UN_viga, canvas_UN_estribos, combo_r, label_r_info, spinbox_r, spinbox_UN_r, ax, fi
7619                 g, base_fig, borde_valor, canvas_tk, margen_grilla

```

```
7608 global estribos_plots,ultimo_estribo,cotas_plots,etiquetas_plots,entry_UN_viga
7609
7610 # Si la subventana ya fue abierta, no hacer nada
7611 if subventana_abierta_cortante:
7612     return
7613
7614 # Crear una nueva ventana para el refuerzo (subventana)
7615 subventana_refuerzo = tk.Toplevel() # Usamos Toplevel() para crear una
subventana independiente
7616
7617 # Aseguramos que la subventana esté siempre en el frente
7618 subventana_refuerzo.attributes('-topmost', True)
7619
7620 # Configuración de la subventana
7621 subventana_refuerzo.geometry('800x400')
7622 subventana_refuerzo.title("Configuración de refuerzo transversal")
7623 subventana_refuerzo.resizable(False, False)
7624
7625 # Crear contenido
7626
7627 labelframe_sr = tk.LabelFrame(subventana_refuerzo,text="")
7628 labelframe_sr.place(x=10,y=10,width=780,height=340)
7629
7630 labelframe_sr_1 = tk.LabelFrame(labelframe_sr,text="General")
7631 labelframe_sr_1.place(x=10,y=10,width=350,height=60)
7632
7633 label_estribo_sr = tk.Label(labelframe_sr_1,text="Estribo:")
7634 label_estribo_sr.place(x=10,y=10)
7635
7636 combobox_estribo_st = ttk.Combobox(labelframe_sr_1, values=["E#3", "E#4"])
7637 combobox_estribo_st.place(x=70,y=10,width=100)
7638 combobox_estribo_st.bind("<<ComboboxSelected>>", actualizar_diametro)
7639
7640 label_diametro_sr = tk.Label(labelframe_sr_1,text="Øbarra:")
7641 label_diametro_sr.place(x=200,y=10)
7642
7643 entry_diametro_sr = tk.Entry(labelframe_sr_1)
7644 entry_diametro_sr.place(x=250,y=10,width=80)
7645
7646 labelframe_control = tk.LabelFrame(labelframe_sr,text="Control de separación
de estribos")
7647 labelframe_control.place(x=10,y=70,width=350,height=250)
7648
7649 # Crear la tabla Treeview en la pestaña 'tad_optimización'
7650 treeview_control = ttk.Treeview(labelframe_control, columns=("L.refuerzo",
"L(m)", "S(m)", "UN"), show="headings")
7651 treeview_control.place(x=5, y=10, width=250, height=100)
7652
7653 # Definir las columnas
7654 treeview_control.heading("L.refuerzo", text="L.refuerzo")
7655 treeview_control.heading("L(m)", text="L(m)")
7656 treeview_control.heading("S(m)", text="S(m)")
7657 treeview_control.heading("UN", text="UN")
7658
7659 # Configurar el ancho de las columnas
7660 treeview_control.column("L.refuerzo", width=40,anchor="center")
7661 treeview_control.column("L(m)", width=20,anchor="center")
7662 treeview_control.column("S(m)", width=20,anchor="center")
7663 treeview_control.column("UN", width=20,anchor="center")
7664
```

```
7665 # Insertar las filas con los datos en la primera columna y vacío en el resto
7666 treeview_control.insert("", "end", values=("Tramo 1", "", "", ""))
7667 treeview_control.insert("", "end", values=("Tramo 2", "", "", ""))
7668 treeview_control.insert("", "end", values=("Tramo 3", "", "", ""))
7669
7670 def no_resize(event):
7671     # Restaurar el tamaño original de las columnas usando un ciclo
7672     for col in treeview_control["columns"]:
7673         treeview_control.column(col, width=62, anchor="center")
7674
7675     # Vincular el evento de clic con la función no_resize
7676 treeview_control.bind("<ButtonRelease-1>", no_resize)
7677
7678 boton_agregar_control =
7679 ttk.Button(labelframe_control, text="Agregar", command=abrir_cuadro_agregar_control)
7680 boton_agregar_control.place(x=260, y=20, width=80)
7681
7682 boton_modificar_control =
7683 ttk.Button(labelframe_control, text="Modificar", command=modificar_fila_control)
7684 boton_modificar_control.place(x=260, y=60, width=80)
7685
7686 labelframe_UN_Flejes = tk.LabelFrame(labelframe_control, text="Unidad de
7687 estribos")
7688 labelframe_UN_Flejes.place(x=5, y=120, width=150, height=100)
7689
7690 var_UN_estribos = tk.IntVar(value=0)
7691 radiobutton_UN_estribos = tk.Radiobutton(labelframe_UN_Flejes, text="Cantidad
7692 estribos", variable=var_UN_estribos,
7693 command=Cantidad_estribos)
7694 radiobutton_UN_estribos.place(x=1, y=10)
7695
7696 entry_UN_flejes = tk.Entry(labelframe_UN_Flejes)
7697 entry_UN_flejes.place(x=10, y=40, width=100)
7698
7699 labelframe_UN_seccion = tk.LabelFrame(labelframe_control, text="Sección")
7700 labelframe_UN_seccion.place(x=170, y=120, width=170, height=100)
7701
7702 label_UN_viga = tk.Label(labelframe_UN_seccion, text="Viga:")
7703 label_UN_viga.place(x=10, y=10)
7704
7705 entry_UN_viga = tk.Entry(labelframe_UN_seccion)
7706 entry_UN_viga.place(x=50, y=10, width=100)
7707
7708 label_UN_luz = tk.Label(labelframe_UN_seccion, text="Luz:")
7709 label_UN_luz.place(x=10, y=50)
7710
7711 entry_UN_luz = tk.Entry(labelframe_UN_seccion)
7712 entry_UN_luz.place(x=50, y=50, width=100)
7713
7714 frame_UN_luz = tk.Frame(labelframe_sr, bd=2, relief="ridge")
7715 frame_UN_luz.place(x=370, y=17, width=395, height=180)
7716
7717 Notebook_UN_estribos =
7718 ttk.Notebook(frame_UN_luz, style="CustomNotebook.TNotebook")
7719
7720 tab_UN_estribos = ttk.Frame(Notebook_UN_estribos)
```

```
7719     Notebook_UN_estribos.add(tab_UN_estribos, text="Despiece refuerzo
transversal")
7720
7721     Notebook_UN_estribos.pack(fill="both", expand=True)
7722
7723     canvas_UN_estribos = tk.Canvas(tab_UN_estribos,bg="white")
7724     canvas_UN_estribos.pack(fill="both", expand=True)
7725
7726     # Función para redimensionar la imagen
7727     def redimensionar_imagen(ruta_imagen, tamaño):
7728         imagen_lista_UN = Image.open(ruta_imagen)
7729         imagen_lista_UN = imagen_lista_UN.resize(tamaño)
7730         imagen_generar_UN = Image.open(ruta_imagen)
7731         imagen_generar_UN = imagen_lista_UN.resize(tamaño)
7732         return ImageTk.PhotoImage(imagen_lista_UN,imagen_generar_UN )
7733
7734     # Cargar las imágenes de los iconos redimensionadas
7735     imagen_dibujar_UN =
redimensionar_imagen("Iconos_proyecto/barra_desplegable.png", (15, 15))
7736     imagen_despiece_UN =
redimensionar_imagen("Iconos_proyecto/generar_imagen.png", (15, 15))
7737
7738     style = ttk.Style()
7739     style.configure("Custom.TButton", background="lightgrey", relief="flat")
7740
7741     boton_dibujar_UN =
tk.Button(labelframe_sr,image=imagen_dibujar_UN,style="Custom.TButton")
7742     boton_dibujar_UN .place(x=580,y=19)
7743
7744     boton_generar_UN =
tk.Button(labelframe_sr,image=imagen_despiece_UN,style="Custom.TButton",command=g
uardar_imagen_dpi)
7745     boton_generar_UN .place(x=640,y=19)
7746
7747     # Mantener una referencia de las imágenes
7748     boton_dibujar_UN.icono = imagen_dibujar_UN
7749     boton_generar_UN.icono = imagen_despiece_UN
7750
7751     # Crear el menú desplegable
7752     menu_opciones = tk.Menu(labelframe_sr, tearoff=0)
7753     menu_opciones.add_command(label="Dibujar esquema", command=lambda:
seleccionar_opcion("Dibujar esquema"))
7754
7755
7756     boton_dibujar_UN.bind("<Button-1>", mostrar_menu)
7757
7758     labelframe_configuraciones = tk.LabelFrame(labelframe_sr,text="Ajuste refuerzo
transversal")
7759     labelframe_configuraciones.place(x=370,y=200,width=395,height=120)
7760
7761     label_r = tk.Label(labelframe_configuraciones,text="L.refuerzo:")
7762     label_r.place(x=5,y=10)
7763
7764     combo_r = ttk.Combobox(labelframe_configuraciones, values=["Tramo 1", "Tramo
2", "Tramo 3"])
7765     combo_r.place(x=70 ,y=10, width=100)
7766     combo_r.bind("<<ComboboxSelected>>", actualizar_label_con_datos)
7767
7768     label_r_info =
tk.Label(labelframe_configuraciones,text="",bd=2,relief="sunken",
```

```
7769         justify="left",
7770         anchor="center",
7771         wraplength=320)
7772     label_r_info .place(x=5,y=35,width=170,height=60)
7773
7774     label_S_m = tk.Label(labelframe_configuraciones,text="S(m):")
7775     label_S_m.place(x=230,y=10)
7776
7777     # Crear un Spinbox
7778     spinbox_r = tk.Spinbox(labelframe_configuraciones, from_=0.00,
7779 to=100.00,format="%.2f", increment=0.01)
7780     spinbox_r.place(x=280,y=10,width=70)
7781
7782     label_UN_m = tk.Label(labelframe_configuraciones,text="UN:")
7783     label_UN_m.place(x=230,y=40)
7784
7785     # Crear un Spinbox
7786     spinbox_UN_r = tk.Spinbox(labelframe_configuraciones, from_=1.0,
7787 to=100.00,format="%.1f", increment=1.0)
7788     spinbox_UN_r.place(x=280,y=40,width=70)
7789     spinbox_UN_r.config(command=dibujar_estribos)
7790
7791     boton_Un_aceptar = ttk.Button(subventana_refuerzo,text="Aceptar")
7792     boton_Un_aceptar.place(x=140,y=360,width=100)
7793
7794     boton_Un_agregar = ttk.Button(subventana_refuerzo,text="Agregar datos")
7795     boton_Un_agregar.place(x=330,y=360,width=100)
7796
7797     boton_Un_cancelar =
7798     ttk.Button(subventana_refuerzo,text="Cancelar",command=lambda:
7799 cerrar_subventana_refuerzo(subventana_refuerzo))
7800     boton_Un_cancelar.place(x=530,y=360,width=100)
7801
7802     # Vincular Shift + Enter para borrar solo los estribos
7803     subventana_refuerzo.bind("<Shift-Return>", borrar_estribos)
7804
7805     # Marcar que la subventana ha sido abierta
7806     subventana_abierta_cortante = True
7807
7808     # Configurar el evento de cierre de la subventana
7809     subventana_refuerzo.protocol("WM_DELETE_WINDOW", lambda:
7810 cerrar_subventana_refuerzo(subventana_refuerzo))
7811
7812     # Función para manejar el cierre de la subventana y permitir abrirla de nuevo
7813     def cerrar_subventana_refuerzo(subventana_refuerzo):
7814         global subventana_abierta_cortante
7815
7816         # Restablecer el estado para permitir abrir la subventana de nuevo
7817         subventana_abierta_cortante = False
7818
7819         # Cerrar la subventana
7820         subventana_refuerzo.destroy()
7821
7822     # Función para mostrar el menú en la posición del botón
7823     def mostrar_menu(event):
7824         menu_opciones.post(event.x_root, event.y_root)
7825
7826     def extraer_valor_numerico(texto):
7827         match = re.search(r"\d+(\.\d+)?", texto) # Busca números en el texto
```

```
7824     return float(match.group()) if match else 1.0 # Retorna el número o 1.0 por
defecto
7825
7826 def dibujar_esquema():
7827     global ventana_dibujar, fig, ax
, base_fig, borde_valor, canvas_tk, margen_grilla, estribos_plots, ultimo_estribo, cotas_
plots, etiquetas_plots
7828
7829
7830     # Obtener valor numérico de la base desde entry_UN_luz
7831     texto_luz = entry_UN_luz.get()
7832     base_fig = extraer_valor_numerico(texto_luz)
7833     # Obtener el valor numérico de entry_borde_C
7834     texto_borde = entry_borde_C.get()
7835     borde_valor = extraer_valor_numerico(texto_borde)
7836     altura_fig = 1.0
7837     texto_viga = entry_UN_viga.get()
7838     texto_estribo = combobox_estribo_st.get()
7839
7840
7841     # Crear la figura y el eje
7842     fig, ax = plt.subplots(figsize=(4, 2))
7843     ax.set_xlim(0, base_fig)
7844     ax.set_ylim(-1, 1)
7845
7846
7847     # Dibujar rectángulo
7848     rect = plt.Rectangle((0, -0.5), base_fig, 1, edgecolor='black',
facecolor='lightgrey')
7849     ax.add_patch(rect)
7850
7851     # Dibujar línea discontinua roja según el valor de borde
7852     if 0 < borde_valor < base_fig:
7853         ax.plot([borde_valor, borde_valor], [-0.5, 0.5], color='red', linestyle='-
-', linewidth=1.5, label="Borde C")
7854
7855     # Crear la línea discontinua desplazada a la izquierda
7856     borde_copia = base_fig - borde_valor # Nueva posición desplazada
7857     ax.plot([borde_copia, borde_copia], [-0.5, 0.5], color='red', linestyle='-
-', linewidth=1.5, label="Borde Copia")
7858
7859     # Líneas de grilla a 0.05 m de cada borde
7860     margen_grilla = 0.05
7861
7862     # Grilla a la derecha de la línea discontinua original
7863     ax.plot([borde_valor + margen_grilla, borde_valor + margen_grilla], [-0.5,
0.5], color='gray', linestyle='--', linewidth=1, label="Grilla")
7864
7865     # Grilla a la izquierda de la línea copia
7866     ax.plot([borde_copia - margen_grilla, borde_copia - margen_grilla], [-0.5,
0.5], color='gray', linestyle='--', linewidth=1, label="Grilla")
7867
7868
7869     # Ocultar títulos de los ejes
7870     ax.set_xlabel("")
7871     ax.set_ylabel("")
7872
7873     # Ocultar números de los ejes
7874     ax.set_xticks([])
7875     ax.set_yticks([])
```

```

7876
7877     # Ajustar color de las líneas de los ejes a gris suave
7878     ax.axhline(0, color='gray', linewidth=1, alpha=0.5)
7879     ax.axvline(0, color='gray', linewidth=1, alpha=0.5)
7880
7881     # Crear líneas para la leyenda
7882     linea_luz, = ax.plot([], [], color="black", linewidth=2, label=f"Luz:
{texto_luz}")
7883     linea_viga, = ax.plot([], [], color="gray", linewidth=2, label=f"Viga:
{texto_viga}")
7884     linea_estribo, = ax.plot([], [], color="black", linewidth=2, label=f"Estribo:
{texto_estribo}")
7885
7886     # Agregar la leyenda en dos columnas, más abajo y con fuente 6
7887     ax.legend(
7888         handles=[linea_luz, linea_viga, linea_estribo],
7889         loc="upper left",
7890         ncol=3,
7891         frameon=False,
7892         prop={'size': 6},
7893         bbox_to_anchor=(0, 0.20)
7894     )
7895
7896     # Limpiar canvas antes de dibujar
7897     for widget in canvas_UN_estribos.wininfo_children():
7898         widget.destroy()
7899
7900     # Insertar gráfico en el canvas de Tkinter
7901     canvas_tk = FigureCanvasTkAgg(fig, master=canvas_UN_estribos)
7902     canvas_tk.get_tk_widget().pack(fill=tk.BOTH, expand=True)
7903     canvas_tk.draw()
7904
7905     # Devolver la figura y el eje para que se puedan usar en otras funciones
7906     return fig, ax
7907
7908
7909 def dibujar_estribos(*args):
7910     global fig, ax, base_fig, borde_valor, canvas_tk, margen_grilla,
estribos_plots, ultimo_estribo
7911     global cotas_plots, etiquetas_plots
7912
7913     try:
7914         # Obtener valores desde los Spinbox y ComboBox
7915         separacion_estribos = float(spinbox_r.get())
7916         cantidad_estribos = int(float(spinbox_UN_r.get()))
7917         longitud_refuerzo = combo_r.get()
7918         tipo_estribo_1 = combobox_estribo_st.get() # Obtener tipo de estribo
7919
7920         # Diccionario de colores por tipo de refuerzo
7921         colores = {"Tramo 1": "blue", "Tramo 2": "green", "Tramo 3": "red"}
7922         color_estribo = colores.get(longitud_refuerzo, "black")
7923
7924         # Posiciones iniciales para la grilla derecha e izquierda
7925         if ultimo_estribo["derecha"] is None:
7926             posicion_actual = borde_valor + margen_grilla # Inicia en la grilla
derecha
7927         else:
7928             posicion_actual = ultimo_estribo["derecha"] + separacion_estribos #
Nuevo inicio
7929

```

```

7930         if ultimo_estribo["izquierda"] is None:
7931             posicion_espejo = base_fig - borde_valor - margen_grilla # Inicia en
la grilla izquierda
7932         else:
7933             posicion_espejo = ultimo_estribo["izquierda"] - separacion_estribos #
Nuevo inicio
7934
7935         # Si es "Tramo 3", se dibujan tres estribos: centro, derecha e izquierda
7936         if longitud_refuerzo == "Tramo 3":
7937             posiciones_especiales = [base_fig / 2,
7938                                     (base_fig / 2) + separacion_estribos,
7939                                     (base_fig / 2) - separacion_estribos]
7940         else:
7941             posiciones_especiales = None
7942
7943         primer_estribo_derecha = None
7944         ultimo_estribo_derecha = None
7945         primer_estribo_izquierda = None
7946         ultimo_estribo_izquierda = None
7947
7948         # Dibujar los estribos
7949         for _ in range(cantidad_estribos):
7950             if posiciones_especiales:
7951                 # Dibujar los tres estribos si es "Tramo 3"
7952                 for pos in posiciones_especiales:
7953                     line, = ax.plot([pos, pos], [-0.15, 0.15],
color=color_estribo, linewidth=1)
7954                     estribos_plots.append((line, color_estribo))
7955
7956                 # Dibujar la cota para "Tramo 3"
7957                 inicio = posiciones_especiales[-1] # Última posición (más a la
izquierda)
7958                 fin = posiciones_especiales[1] # Segunda posición (más a la
derecha)
7959                 dibujar_cota(inicio, fin, "centro") # Nueva cota para Tramo 3
7960
7961                 # Etiqueta debajo del grupo de estribos
7962                 agregar_etiqueta((inicio + fin) / 2, f"{cantidad_estribos}
{tipo_estribo_1}@{separacion_estribos:.2f}")
7963
7964                 # Evitar que siga con la lógica normal
7965                 break
7966
7967         # Estribo en la grilla derecha
7968         line, = ax.plot([posicion_actual, posicion_actual], [-0.15, 0.15],
color=color_estribo, linewidth=1)
7969         estribos_plots.append((line, color_estribo))
7970         if primer_estribo_derecha is None:
7971             primer_estribo_derecha = posicion_actual
7972             ultimo_estribo_derecha = posicion_actual
7973             ultimo_estribo["derecha"] = posicion_actual # Guardar última posición
7974
7975         # Estribo en la grilla izquierda
7976         line_esp, = ax.plot([posicion_espejo, posicion_espejo], [-0.15, 0.15],
color=color_estribo, linewidth=1)
7977         estribos_plots.append((line_esp, color_estribo))
7978         if primer_estribo_izquierda is None:
7979             primer_estribo_izquierda = posicion_espejo
7980             ultimo_estribo_izquierda = posicion_espejo

```

```

7981         ultimo_estribo["izquierda"] = posicion_espejo # Guardar última
posición
7982
7983         # Avanzar la posición según la separación
7984         posicion_actual += separacion_estribos
7985         posicion_espejo -= separacion_estribos
7986
7987         # Dibujar cotas para cada grupo de estribos
7988         if primer_estribo_derecha is not None and ultimo_estribo_derecha is not
None:
7989             dibujar_cota(primer_estribo_derecha, ultimo_estribo_derecha,
"derecha")
7990             agregar_etiqueta((primer_estribo_derecha + ultimo_estribo_derecha) /
2, f"{cantidad_estribos}{tipo_estribo_1}@{separacion_estribos:.2f}")
7991
7992             if primer_estribo_izquierda is not None and ultimo_estribo_izquierda is
not None:
7993                 dibujar_cota(primer_estribo_izquierda, ultimo_estribo_izquierda,
"izquierda")
7994                 agregar_etiqueta((primer_estribo_izquierda + ultimo_estribo_izquierda)
/ 2, f"{cantidad_estribos}{tipo_estribo_1}@{separacion_estribos:.2f}")
7995
7996         # Redibujar la gráfica
7997         canvas_tk.draw()
7998
8000     except Exception as e:
8001         print(f"Error en dibujar_estribos: {e}")
8002
8003
8004 def dibujar_cota(inicio, fin, lado):
8005     """Dibuja una cota con líneas de extensión y etiqueta de distancia"""
8006     global ax, cotas_plots
8007
8008     # Línea horizontal de la cota
8009     y_cota = 0.20 # Un poco arriba de los estribos
8010     linea_cota, = ax.plot([inicio, fin], [y_cota, y_cota], color="black",
linewidth=1)
8011
8012     # Líneas de extensión en los extremos
8013     linea_ext_izq, = ax.plot([inicio, inicio], [y_cota - 0.05, y_cota + 0.05],
color="black", linewidth=1)
8014     linea_ext_der, = ax.plot([fin, fin], [y_cota - 0.05, y_cota + 0.05],
color="black", linewidth=1)
8015
8016     # Etiqueta con la distancia
8017     distancia = abs(fin - inicio)
8018     texto_cota = ax.text((inicio + fin) / 2, y_cota + 0.02, f"{distancia:.2f}m",
ha="center", va="bottom", fontsize=6, color="black")
8019
8020     # Guardar las cotas para poder eliminarlas después
8021     cotas_plots.append((linea_cota, linea_ext_izq, linea_ext_der, texto_cota))
8022
8023
8024
8025 def agregar_etiqueta(x, texto):
8026     """Dibuja una etiqueta de texto debajo del grupo de estribos"""
8027     global ax, etiquetas_plots
8028     etiqueta = ax.text(x, -0.25, texto, ha="center", va="top", fontsize=6,
color="black")
8029     etiquetas_plots.append(etiqueta)

```

```
8030
8031 def borrar_estribos(event=None):
8032     global estribos_plots, ultimo_estribo, cotas_plots, etiquetas_plots
8033
8034
8035     if not estribos_plots:
8036         return # Si no hay estribos, no hace nada
8037
8038     try:
8039         # Eliminar cada línea de estribo de la gráfica
8040         for line, _ in estribos_plots:
8041             line.remove()
8042
8043         # Vaciar la lista después de eliminar los estribos
8044         estribos_plots = []
8045         ultimo_estribo = {"derecha": None, "izquierda": None}
8046
8047         # Eliminar todas las cotas
8048         for linea_cota, linea_ext_izq, linea_ext_der, texto_cota in cotas_plots:
8049             linea_cota.remove()
8050             linea_ext_izq.remove()
8051             linea_ext_der.remove()
8052             texto_cota.remove()
8053         cotas_plots = []
8054
8055
8056         for etiqueta in etiquetas_plots:
8057             etiqueta.remove()
8058         etiquetas_plots = []
8059
8060         canvas_tk.draw()
8061
8062     except Exception as e:
8063         print(f"Error en borrar_estribos: {e}")
8064
8065 def guardar_imagen_dpi():
8066     """Guarda la imagen del gráfico con todos los elementos dibujados."""
8067     try:
8068         ruta_guardado = os.path.join("Reportes", "imagenes_pilot",
8069 "grafico_estribos.png")
8070         fig.savefig(ruta_guardado, dpi=300, bbox_inches="tight")
8071         print(f"Imagen guardada en: {ruta_guardado}")
8072     except Exception as e:
8073         print(f"Error al guardar la imagen: {e}")
8074
8075 def seleccionar_opcion(opcion):
8076     if opcion == "Dibujar esquema":
8077         dibujar_esquema() # Dibuja el esquema
8078
8079
8080
8081 # Variable global para verificar si la subventana de agregar control ya fue
8082 # abierta
8083 ventana_agregar_abierta = False
8084
8085 # Función para abrir el cuadro de agregar control
8086 def abrir_cuadro_agregar_control():
8087     global ventana_agregar_abierta
```

```

8088     # Verificar si ya se abrió una ventana de agregar control
8089     if ventana_agregar_abierta:
8090         return # No hacer nada si ya está abierta
8091
8092     # Crear una nueva subventana
8093     cuadro_agregar_control = tk.Toplevel()
8094     cuadro_agregar_control.geometry("330x200")
8095     cuadro_agregar_control.title("Agregar opciones")
8096     cuadro_agregar_control.resizable(False, False) # Evitar redimensionar la
ventana
8097     cuadro_agregar_control.attributes('-topmost', True) # Mantener la ventana al
frente
8098
8099     # Longitud refuerzo
8100     label_l_refuerzo = tk.Label(cuadro_agregar_control, text="Longitud refuerzo:")
8101     label_l_refuerzo.place(x=10, y=10)
8102
8103     combo_l_refuerzo = ttk.Combobox(cuadro_agregar_control, values=["Tramo 1",
"Tramo 2", "Tramo 3"])
8104     combo_l_refuerzo.place(x=120, y=10, width=170)
8105
8106     # Separación
8107     label_separacion = tk.Label(cuadro_agregar_control, text="Separación [m]:")
8108     label_separacion.place(x=10, y=90)
8109
8110     entry_control_S = tk.Entry(cuadro_agregar_control)
8111     entry_control_S.place(x=120, y=90, width=170)
8112
8113     # Longitud
8114     label_longitud_m = tk.Label(cuadro_agregar_control, text="Longitud [m]:")
8115     label_longitud_m.place(x=10, y=50)
8116
8117     entry_control_longitud = tk.Entry(cuadro_agregar_control)
8118     entry_control_longitud.place(x=120, y=50, width=170)
8119
8120     # Botón para agregar datos
8121     boton_aceptar_control = ttk.Button(cuadro_agregar_control, text="Aceptar",
command=lambda:
agregar_datos_al_treeview(combo_l_refuerzo, entry_control_S,
entry_control_longitud))
8122     boton_aceptar_control.place(x=60, y=140)
8123
8124
8125     # Botón para cerrar
8126     boton_cerrar_control = ttk.Button(cuadro_agregar_control, text="Cancelar",
command=lambda: cerrar_ventana_agregar(cuadro_agregar_control))
8127     boton_cerrar_control.place(x=180, y=140)
8128
8129     # Marcar que la ventana de agregar control está abierta
8130     ventana_agregar_abierta = True
8131
8132     # Configurar el evento de cierre de la ventana
8133     cuadro_agregar_control.protocol("WM_DELETE_WINDOW", lambda:
cerrar_ventana_agregar(cuadro_agregar_control))
8134
8135     # Función para cerrar la ventana de agregar control
8136     def cerrar_ventana_agregar(cuadro):
8137         global ventana_agregar_abierta
8138         ventana_agregar_abierta = False # Actualizar la variable para permitir
abrir la nuevamente
8139         cuadro.destroy() # Cerrar la ventana

```

```
8140
8141 # Función para agregar datos al Treeview
8142 def agregar_datos_al_treeview(combo_l_refuerzo, entry_control_S,
entry_control_longitud):
8143     # Obtener los valores de los Entry y ComboBox
8144     seleccion_refuerzo = combo_l_refuerzo.get()
8145     separacion_r_s = entry_control_S.get()
8146     longitud_r_s = entry_control_longitud.get()
8147
8148     # Asegurarnos de que los valores son numéricos para realizar el cálculo
8149     try:
8150         separacion_r_s = float(separacion_r_s)
8151         longitud_r_s = float(longitud_r_s)
8152     except ValueError:
8153         return # Si los valores no son numéricos, no hacer nada
8154
8155     # Cálculo de UN_flejes
8156     if seleccion_refuerzo == "Tramo 1":
8157         UN_flejes = (longitud_r_s / separacion_r_s) + 1
8158         resultado_UN_fleje = math.ceil(UN_flejes)
8159         resultado_UN_fleje = f'{resultado_UN_fleje:.1f}'
8160     elif seleccion_refuerzo in ["Tramo 2", "Tramo 3"]:
8161         UN_flejes = longitud_r_s / separacion_r_s
8162
8163     # Redondeamos a 10 decimales para evitar errores de precisión flotante
8164     UN_flejes = round(UN_flejes, 10)
8165
8166     # Comprobamos si el resultado es entero
8167     if UN_flejes.is_integer():
8168         resultado_UN_fleje = f'{int(UN_flejes):.1f}' # Convertimos a entero y
mostramos con ".0"
8169     else:
8170         resultado_UN_fleje = f'{UN_flejes:.1f}' # Mantenemos formato con un
decimal
8171
8172     # Verificar que los campos no estén vacíos
8173     if not separacion_r_s or not longitud_r_s:
8174         return # Si algún campo está vacío, no hacemos nada
8175
8176     # Según la opción del ComboBox, actualizamos la fila correspondiente
8177     if seleccion_refuerzo == "Tramo 1":
8178         # Actualizamos la primera fila (L.max)
8179         treeview_control.item(treeview_control.get_children()[0], values=("Tramo
1", longitud_r_s, separacion_r_s, resultado_UN_fleje))
8180     elif seleccion_refuerzo == "Tramo 2":
8181         # Actualizamos la segunda fila (L.medio)
8182         treeview_control.item(treeview_control.get_children()[1], values=("Tramo
2", longitud_r_s, separacion_r_s, resultado_UN_fleje))
8183     elif seleccion_refuerzo == "Tramo 3":
8184         # Actualizamos la tercera fila (L.min)
8185         treeview_control.item(treeview_control.get_children()[2], values=("Tramo
3", longitud_r_s, separacion_r_s, resultado_UN_fleje))
8186     else:
8187         return # Si la selección no es válida, no hacer nada
8188
8189
8190 def Cantidad_estribos():
8191     suma_UN = 0
8192
8193     for item in treeview_control.get_children():
```

```

8194     valores = treeview_control.item(item, "values")
8195
8196     if len(valores) > 3:
8197         valor_UN = valores[3].strip()
8198
8199         try:
8200             valor_UN = float(valor_UN) # Convertimos a float
8201             suma_UN += int(valor_UN)   # Convertimos a int y sumamos
correctamente
8202         except ValueError:
8203             pass # Si hay un error, simplemente lo ignoramos
8204
8205         # Obtener solo el número del combobox
8206         valor_combobox = combobox_estribo_st.get()
8207         numero_estribo = ''.join(filter(str.isdigit, valor_combobox)) # Extraer solo
el número
8208
8209         # Formato del resultado
8210         resultado_UN = f"{suma_UN} UN #{numero_estribo}"
8211
8212         # Mostrar resultado en el entry
8213         entry_UN_flejes.delete(0, tk.END)
8214         entry_UN_flejes.insert(0, resultado_UN)
8215
8216         # Obtener los valores de los labels
8217         valor_luz_UN = label_seccion_luz.cget("text")
8218         valor_seccion_UN = label_seccion_cargas.cget("text")
8219
8220         # Actualizar los entries con los valores de los labels
8221         entry_UN_luz.delete(0, tk.END)
8222         entry_UN_luz.insert(0, f"{valor_luz_UN} m")
8223
8224         entry_UN_viga.delete(0, tk.END)
8225         entry_UN_viga.insert(0, valor_seccion_UN)
8226
8227 # Función para obtener los datos del Treeview según el valor seleccionado
8228 def obtener_datos_de_treeview(valor_seleccionado):
8229     # Buscar las filas del Treeview que coinciden con el valor seleccionado
8230     for item in treeview_control.get_children():
8231         # Obtener el valor de la primera columna
8232         valor_fila = treeview_control.item(item, "values")[0]
8233         # Verificar si el valor de la primera columna coincide con la selección
8234         if valor_fila == valor_seleccionado:
8235             # Devolver los valores de la fila
8236             return treeview_control.item(item, "values")
8237     return None # En caso de no encontrar el valor
8238
8239 # Función que se llama cuando el usuario selecciona una opción en el ComboBox
8240 def actualizar_label_con_datos(event):
8241     # Obtener el valor seleccionado del ComboBox
8242     seleccion_r = combo_r.get()
8243
8244     # Obtener los datos del Treeview correspondientes a la opción seleccionada
8245     datos_r = obtener_datos_de_treeview(seleccion_r)
8246
8247     if datos_r:
8248         # Actualizar el texto del Label con los datos de la fila
8249         label_r_info.config(text=f"Datos para {seleccion_r}:\n L(m):{datos_r[1]}\n
S(m):{datos_r[2]}\n UN:{datos_r[3]}", anchor="w", justify="left")
8250     else:

```

```

8251         # En caso de no encontrar los datos, mostrar un mensaje adecuado
8252         label_r_info.config(text=f"No se encontraron datos para {seleccion_r}.")
8253
8254
8255 # Variable global para controlar si la ventana de edición está abierta
8256 ventana_edicion_abierta = None
8257
8258 def modificar_fila_control():
8259     """Permite modificar una fila seleccionada del Treeview."""
8260     global ventana_edicion_abierta
8261
8262     # Verificar si ya hay una ventana de edición abierta
8263     if ventana_edicion_abierta is not None and
ventana_edicion_abierta.winfo_exists():
8264         return # No permitir abrir más de una ventana de edición
8265
8266     selected_item = treeview_control.selection() # Obtener la fila seleccionada
8267
8268     if not selected_item:
8269         return # Si no hay fila seleccionada, no hacer nada
8270
8271     valores_treeview_control = treeview_control.item(selected_item, "values") #
Obtener los valores de la fila
8272
8273     # Crear la ventana de edición
8274     ventana_edicion_abierta = tk.Toplevel()
8275     ventana_edicion_abierta.title("Modificar Fila")
8276     ventana_edicion_abierta.geometry("250x190")
8277     ventana_edicion_abierta.resizable(False, False) # No permitir cambiar tamaño
8278     ventana_edicion_abierta.attributes('-topmost', True) # Mantener ventana al
frente
8279
8280     # Función para cerrar la ventana correctamente
8281     def cerrar_ventana():
8282         global ventana_edicion_abierta
8283         ventana_edicion_abierta.destroy()
8284         ventana_edicion_abierta = None # Marcar como cerrada
8285
8286     ventana_edicion_abierta.protocol("WM_DELETE_WINDOW", cerrar_ventana) # Evento
al cerrar
8287
8288     # Etiquetas y cuadros de entrada
8289     labels_treeview_control = ["L.refuerzo", "L(m)", "S(m)", "UN"]
8290     entries_treeview_control = []
8291
8292     for i, label in enumerate(labels_treeview_control):
8293         tk.Label(ventana_edicion_abierta, text=label).grid(row=i, column=0,
padx=10, pady=5)
8294         entry_treeview_control = tk.Entry(ventana_edicion_abierta)
8295         entry_treeview_control.grid(row=i, column=1, padx=10, pady=5)
8296         entry_treeview_control.insert(0, valores_treeview_control[i]) # Insertar
el valor actual en el Entry
8297         entries_treeview_control.append(entry_treeview_control)
8298
8299     def guardar_cambios():
8300         """Guarda los cambios en el Treeview y cierra la ventana."""
8301         nuevos_valores = [entry.get() for entry in entries_treeview_control]
8302         treeview_control.item(selected_item, values=nuevos_valores) # Actualizar
Treeview
8303         cerrar_ventana() # Cerrar la ventana de edición

```

```

8304
8305     # Botón para guardar cambios
8306     btn_guardar = ttk.Button(ventana_edicion_abierta, text="Guardar",
command=guardar_cambios)
8307     btn_guardar.place(x=90,y=140)
8308
8309
8310 # Función para actualizar el Entry según la selección del ComboBox
8311 def actualizar_diametro(event):
8312     seleccion = combobox_estribo_st.get()
8313     if seleccion == "E#3":
8314         entry_diametro_sr.delete(0, tk.END)
8315         entry_diametro_sr.insert(0, "9.5 mm")
8316     elif seleccion == "E#4":
8317         entry_diametro_sr.delete(0, tk.END)
8318         entry_diametro_sr.insert(0, "12.7 mm")
8319
8320 ventana_abierta = False
8321
8322 def abrir_ventana_cortante():
8323     global
ventana_abierta,frame_V,labelframe_grafica,entry_info_estribo,entry_info_area,comb
o_Vu_Wu,entry_Vu_ultimo
8324     global combo_φ_revision
,combo_fc_revision,combo_factor_λ,entry_φVc_revision,entry_φVcmedios_revision,che
ck_revisar_widget
8325     global
checkboxbutton_1_V,checkboxbutton_2_V,checkboxbutton_3_V,check_condición_1,check_condición
_2,check_condición_3
8326     global
label_mensaje_condicion,entry_resistencia_Vs,label_copia_Vs,label_copia_seccion,la
bel_cumple_Vs,label_conclusion_V
8327     global
entry_Smax,radiobutton_S,entry_Av,radiobutton_Av,label_b_prueba,radiobutton_ramas,
entry_ramas,label_ramas_V,entry_Vs_estribos
8328     global
label_Vs_estribos_1,entry_resistencia_Vs_requerida,treeview_cortante,label_resiste
ncia_total,treeview_cortante_valores
8329     global entry_ancho_C ,
entry_borde_C,check_longitud_refuerzo,check_longitud_refuerzo_var,canvas_longitud,
slider_cortante,labelframe_estribos
8330     global
entry_posicion,imagen_agregar,imagen_eliminar,boton_agregar_marcador,boton_elimina
r_marcador,lineas_discontinuas,cotas,label_primer_estribo
8331     global ultimo_valor_borde_C,textos_cotas,borde_valor,smax_var
8332
8333     # Solo abrir la ventana si no está abierta ya
8334     if not ventana_abierta:
8335         # Crear una nueva ventana de tipo Toplevel
8336         ventana_cortante = tk.Toplevel()
8337
8338         # Configurar la ventana para que no se pueda redimensionar
8339         ventana_cortante.resizable(False, False)
8340
8341         # Asegurar que la ventana esté siempre en el frente
8342         ventana_cortante.attributes('-topmost', True)
8343
8344         # Establecer un tamaño para la ventana
8345         ventana_cortante.geometry('850x600')
8346

```

```
8347 # Agregar un título a la ventana
8348 ventana_cortante.title("Diseño cortante")
8349
8350 # Agregar contenido a la ventana
8351 labelframe_cortante = tk.LabelFrame(ventana_cortante, text="")
8352 labelframe_cortante.place(x=10, y=10, width=830, height=540)
8353
8354 # Crear el estilo para personalizar las pestañas
8355 style = ttk.Style()
8356
8357 # Estilo para el Notebook
8358 style.configure("CustomNotebook.TNotebook",
8359                 background="lightgray",
8360                 tabmargins=[0, 0, 0, 0])
8361
8362 # Estilo para las pestañas
8363 style.configure("CustomNotebook.TNotebook.Tab",
8364                 background="white",
8365                 foreground="black",
8366                 padding=[10, 2],
8367                 font=("Arial", 9))
8368
8369 # Estilo para la pestaña seleccionada
8370 style.map("CustomNotebook.TNotebook.Tab",
8371          background=[("selected", "lightgray")],
8372          foreground=[("selected", "black")])
8373
8374 # Crear el Notebook dentro del LabelFrame
8375 notebook_Cortante =
8376 ttk.Notebook(labelframe_cortante, style="CustomNotebook.TNotebook")
8377 notebook_Cortante.pack(fill="both", expand=True)
8378
8379 # Crear las pestañas del Notebook
8380 tab_requisitos = tk.Frame(notebook_Cortante)
8381 tab_diseño = tk.Frame(notebook_Cortante)
8382 tad_optimización = tk.Frame(notebook_Cortante)
8383
8384 # Insertar las pestañas en el Notebook
8385 notebook_Cortante.add(tab_requisitos, text="Requisitos NSR-10")
8386 notebook_Cortante.add(tab_diseño, text="Diseño cortante")
8387 notebook_Cortante.add(tad_optimización, text="Optimización de estribos")
8388
8389 # Crear labelframe para contenido de generalidades
8390 labelframe_generalidades = tk.LabelFrame(tab_requisitos, text="")
8391 labelframe_generalidades.place(x=10, y=10, width=400, height=490)
8392
8393 # Crear el Notebook dentro del LabelFrame de generalidades
8394 notebook_generalidades =
8395 ttk.Notebook(labelframe_generalidades, style="CustomNotebook.TNotebook")
8396 notebook_generalidades.pack(fill="both", expand=True)
8397
8398 # Crear la pestaña del Notebook de generalidades
8399 tab_generalidades = tk.Frame(notebook_generalidades)
8400
8401 # Crear un Canvas dentro de la pestaña
8402 canvas_cortante = tk.Canvas(tab_generalidades, width=380, height=460)
8403 canvas_cortante.pack(fill="both", expand=True)
8404
8405 labelframe_V= tk.LabelFrame(canvas_cortante, text="")
```

```

8405
8406     label_V = tk.Label(labelframe_V, text="El diseño a cortante es fundamental
porque la falla asociada a este efecto es considerada una falla frágil. Este tipo
de falla ocurre de manera repentina y sin señales previas, lo cual es muy
riesgoso. Por ello, en el diseño estructural de una viga, se busca evitar la falla
por cortante.",
8407         justify="left", anchor="center", wraplength=320 )
8408     label_V.place(x=0,y=10)
8409
8410     frame_V = tk.Frame(canvas_cortante,bg="white",bd=2,relief="ridge")
8411
8412     # Cargar la imagen
8413     img_path = "Iconos_proyecto/esfuerzos.png" # Asegúrate de que la ruta sea
correcta
8414     img = Image.open(img_path)
8415     img = img.resize((300, 100)) # Ajusta el tamaño de la imagen según sea
necesario
8416
8417     # Convierte la imagen a un formato que Tkinter pueda usar
8418     img_tk = ImageTk.PhotoImage(img)
8419
8420     # Inserta la imagen en un Label dentro del Frame
8421     label_img = tk.Label(frame_V, image=img_tk)
8422     label_img.image = img_tk # Mantén una referencia de la imagen
8423     label_img.pack()
8424
8425     labelframe_V_info = tk.LabelFrame(canvas_cortante)
8426     label_V_info = tk.Label(labelframe_V_info,
8427         text="En una sección transversal, los esfuerzos son una
combinación de esfuerzos cortantes y normales. En el eje neutro, el esfuerzo
cortante alcanza su valor máximo, mientras que el esfuerzo normal es cero. Esta
condición permite que los esfuerzos principales se ubiquen en un plano de 45°.",
8428         justify="left", anchor="center", wraplength=320)
8429     label_V_info.place(x=0,y=0)
8430
8431     labelframe_V_falla = tk.LabelFrame(canvas_cortante,text="Falla a
cortante")
8432     label_V_falla= tk.Label(labelframe_V_falla,text="En el diseño a cortante
de vigas de concreto reforzado, se busca que la viga falle por flexión, lo que es
una falla dúctil, en lugar de fallar por cortante, que es una falla frágil. Esta
última se considera frágil porque genera tensiones diagonales.",
8433         justify="left", anchor="center", wraplength=320)
8434     label_V_falla.place(x=0,y=0)
8435
8436     frame_V_falla = tk.Frame(canvas_cortante,bg="white",bd=2,relief="ridge")
8437
8438     # Cargar la imagen
8439     img_cortante = Image.open("Iconos_proyecto/viga_reforzada.png") # Ruta de
la imagen
8440     img_cortante = img_cortante.resize((150, 100)) # Puedes ajustar el tamaño
de la imagen si es necesario
8441     photo_cortante = ImageTk.PhotoImage(img_cortante)
8442
8443     # Crear un Label para contener la imagen
8444     label_img_cortante = tk.Label(frame_V_falla, image=photo_cortante,
bg="white")
8445     label_img_cortante.image = photo_cortante # Mantén una referencia de la
imagen
8446     label_img_cortante.pack()
8447

```

```

8448     labelframe_V_1 = tk.LabelFrame(canvas_cortante,text="")
8449
8450     label_V_1 = tk.Label(
8451         labelframe_V_1,
8452         text="El elemento que contrarrestará la falla por cortante son los
estribos. Lo ideal es colocar estribos perpendiculares al eje de la viga y
ubicarlos a una separación mínima necesaria para reducir el efecto de la falla por
tracción diagonal.\n\n"
8453         "El comportamiento de las vigas de concreto reforzado con refuerzo
transversal depende de la carga distribuida a lo largo de toda la longitud de la
viga. Los estribos tienen un área aferente que corresponde a la separación entre
ellos, "
8454         "y la cortante que puede ser absorbida por los estribos está relacionada
con su área. La cortante disminuye a lo largo de la cara de la sección, por lo que
el estribo debe contar con al menos dos ramas de confinamiento para ofrecer
resistencia al cortante.",
8455         justify="left",
8456         anchor="center",
8457         wraplength=320
8458     )
8459     label_V_1.place(x=0,y=0)
8460
8461     # Insertar la pestaña en el Notebook de generalidades
8462     notebook_generalidades.add(tab_generalidades, text="Generalidades")
8463
8464     labelframe_re_V = tk.LabelFrame(tab_requisitos,text="")
8465     labelframe_re_V.place(x=420,y=10,width=390,height=80)
8466
8467     label_re_V = tk.Label(labelframe_re_V,text="A continuación, se detallan
los requisitos mínimos establecidos por la Norma NSR-10.Estos requisitos son
fundamentales para calcular el diseño a cortante.",
8468         justify="left",
8469         anchor="center",
8470         wraplength=350)
8471
8472     label_re_V.place(x=10,y=10)
8473
8474     labelframe_NSR10 = tk.LabelFrame(tab_requisitos,text="")
8475     labelframe_NSR10.place(x=420,y=100,width=390,height=400)
8476
8477     # Crear el Notebook
8478     Notebook_NSR10 =
ttk.Notebook(labelframe_NSR10,style="CustomNotebook.TNotebook") # Crear el
Notebook dentro del LabelFrame
8479
8480     # Crear una pestaña dentro del Notebook
8481     tab_requisitos_NSR10 = ttk.Frame(Notebook_NSR10) # Aquí es donde puedes
colocar el contenido de la pestaña
8482
8483     # Agregar la pestaña al Notebook con el título 'Requisitos'
8484     Notebook_NSR10.add(tab_requisitos_NSR10, text="Requisitos")
8485
8486     # Empaquetar el Notebook en el LabelFrame
8487     Notebook_NSR10.pack(fill="both", expand=True)
8488
8489     # Crear un Canvas dentro de la pestaña 'tab_requisitos_NSR10'
8490     canvas_nsr10 = tk.Canvas(tab_requisitos_NSR10, width=380, height=380) #
Ajusta el tamaño según lo necesario
8491     canvas_nsr10.pack(fill="both", expand=True)
8492

```

```

8493     labelframe_req_1 = tk.LabelFrame(canvas_nsr10,text="")
8494
8495     label_req_1 = tk.Label(labelframe_req_1 ,text="Diseño a Cortante y Torsión
según la Norma NSR-10 (Capítulo C.11).",
8496                             justify="left",
8497                             anchor="center",
8498                             wraplength=280)
8499
8500     label_req_1.place(x=0,y=10)
8501
8502     labelframe_req_2 = tk.LabelFrame(canvas_nsr10,text="")
8503
8504     label_req_2 = tk.Label(labelframe_req_2, text="[C.11.1] Se define el
metodo de resistencia a cortante.\n\n"
8505                             "El método de resistencia establece como condición de diseño que
la solicitación a la que está sometido un elemento estructural debe ser menor o
igual a su capacidad de resistencia.\n\n"
8506                             "Vu = Resistencia o solicitación requerida\n"
8507                             " $\phi V_n$  = Capacidad nominal de diseño\n\n"
8508                             "Especifica tambien que la resistencia nominal de diseño va ser
igual a lo que aporta el concreto más el acero, Por lo tanto, la expresión
será:\n\n"
8509                             "Vu =  $\phi[V_c + V_s]$ \n\n"
8510                             "[C.9.3.2.3] Se define el valor de  $\phi$  para el diseño a cortante y
torsión. En este caso,  $\phi$  para el diseño cortante se establece en 0.75, ya que la
norma reduce la capacidad de diseño con el objetivo de garantizar que el elemento
falle primero por flexión antes que por cortante.\n\n"
8511                             "[C.11.1.3] Especifica las condiciones bajo las cuales se puede
realizar el diseño de cortante [Vu].\n\n",
8512                             justify="left",
8513                             anchor="center",
8514                             wraplength=300)
8515
8516     label_req_2.place(x=10,y=10)
8517
8518     frame_req_2 = tk.Frame(labelframe_req_2,bd=2,relief="ridge")
8519     frame_req_2.place(x=10,y=410,width=300,height=300)
8520
8521     # Ruta de la imagen (cambiar a 'image_path_req')
8522     image_path_req = "Iconos_proyecto/diagrama_V_Wu.png" # Ruta relativa
8523
8524     # Abrir la imagen con PIL
8525     img_pil = Image.open(image_path_req)
8526
8527     # Redimensionar la imagen (ajustar al tamaño del frame)
8528     img_resized = img_pil.resize((280, 280)) # Cambiar 300x300 por las
dimensiones que prefieras
8529
8530     # Convertir la imagen redimensionada a un formato compatible con Tkinter
8531     img_req = ImageTk.PhotoImage(img_resized)
8532
8533     # Insertar la imagen en el Frame con un Label
8534     label_with_image = tk.Label(frame_req_2, image=img_req,bg="white")
8535     label_with_image.pack(fill="both", expand=True)
8536
8537     # Mantener la referencia de la imagen
8538     label_with_image.image = img_req
8539
8540     label_req_3 = tk.Label(labelframe_req_2,text="Vmax = Cortante máximo en el
eje del apoyo.\n"

```

```

8541         "Vud = Cortante a una distancia 'd',de la cara del
apoyo\n\n"
8542         "Como evidentemente Vud es menor que Vmax, la norma
permite diseñar la viga con Vud."
8543         "Si se cumple que inicialmente existe una reacción de
compresión,la siguiente condición es que entre la cara del apoyo y la distancia
'd' no debe haber una carga puntual.La carga puntual debe ubicarse más allá de
dicha distancia 'd'.\n\n"
8544         "[C.11.2] Se menciona la resistencia a cortante
proporcionada por el concreto [Vc]\n\n"
8545         "[C.11.2.1] La norma especifica el valor de Vc que se
utiliza para calcular la resistencia de los elementos sometidos a flexión y
cortante en vigas\n\n"
8546         "Vc = 0.17*λ*√f'c*b*d\n\n"
8547         "[C.11.2.1] Resistencia a cortante del refuerzo
transversal.\n"
8548         "Tipos: Estribos inclinados.\n"
8549         "         Estribos paralelos al eje de la viga.\n"
8550         "         Estribos espirales.\n\n"
8551         "[C.11.2.1] Limites para la separación a cortante.\n"
8552         "Estribos paralelos al eje:\n"
8553         " A)      Sí Vs < 0.33*√f'c*b*d\n"
8554         "         S ≤ {d/2 ó 60cm}\n"
8555         " B)      Sí Vs > 0.33*√f'c*b*d\n"
8556         "         S ≤ {d/4 ó 30cm}\n\n"
8557         "[C.11.4.6] Refuerzo mínimo a cortante.\n"
8558         "Si Vu > φVc Entonces: Diseñar refuerzo transversal\n"
8559         "Si 1/2*φVc < Vu < φVc Entonces: Colocar área
mínima\n"
8560         "Si Vu < 1/2*φVc Entonces: No se necesita refuerzo
transversal\n\n"
8561         "Nota: Estas son las condiciones basicas de como
debemos diseñar una sección de concreto reforzado para cortante."
8562         "El refuerzo transversal se debe llevar por lo menos
hasta una sección donde Vu= 1/2*φVc\n\n"
8563         "[C.11.4.6] La norma presenta una ecuación en función
a una separación mínima.\n"
8564         "A)      S1 = Av*fy/0.062*√f'c*b\n"
8565         "B)      S2 = Av*fy/0.35*b\n"
8566         "C)      Smax = min{S1,S2}\n\n"
8567         "En el diseño a cortante de elementos estructurales,
es mucho más facil garantizar el control de la separación mínima de los estribos
utilizando la ecuación que define la cuantía de acero mínimo.\n\n"
8568         "[C.11.4.7.2] Diseño de refuerzo para cortante Vu >
φVc\n"
8569         "De acuerdo con la norma, los estribos deben colocarse
perpendiculares al eje del elemento. En este sentido, la norma establece que la
capacidad de los estribos será igual a la expresión siguiente:\n\n"
8570         "Vs = d*[Av*fy]/s\n"
8571         "Av = Es la suma del área del número de ramas del
estribo\n",
8572         justify="left",
8573         anchor="center",
8574         wraplength=300)
8575         label_req_3.place(x=10,y=720)
8576
8577         frame_req_3 = tk.Frame(labelframe_req_2,bd=2,relief="ridge",bg="white")
8578         frame_req_3.place(x=80,y=1640,width=170,height=150)
8579
8580         # Ruta de la imagen

```

```

8581     image_path_cortante = "Iconos_proyecto/Cortante_Estribo.png"
8582
8583     # Cargar la imagen con PIL
8584     img_cortante = Image.open(image_path_cortante)
8585
8586     # Redimensionar la imagen si es necesario
8587     img_cortante = img_cortante.resize((170, 150), Image.Resampling.LANCZOS)
8588     # Ajustar al tamaño del Frame
8589
8590     # Convertir la imagen a formato compatible con Tkinter
8591     photo_cortante = ImageTk.PhotoImage(img_cortante)
8592
8593     # Crear un Label para mostrar la imagen dentro del Frame
8594     label_img_cortante = tk.Label(frame_req_3, image=photo_cortante)
8595     label_img_cortante.image = photo_cortante # Guardar la referencia de la
imagen
8596     label_img_cortante.pack()
8597
8598     label_req_4= tk.Label(labelframe_req_2,text="[C.11.4.7.9] Establece que se
debe verificar el control de tamaño de la sección:\n\n"
8599                        "A)  $V_s \leq 0.66 \cdot \sqrt{f'_c \cdot b \cdot d}$ \n"
8600                        "A) Si  $V_s > 0.66 \cdot \sqrt{f'_c \cdot b \cdot d}$  entonces, se debe aumentar
las dimensiones de la sección.\n\n"
                        "Finalmente, estos serían los requisitos mínimos para
el cálculo del diseño a cortante en vigas, ahora realiza las configuraciones
correspondientes",
8601                        justify="left",
8602                        anchor="center",
8603                        wraplength=300)
8604     label_req_4.place(x=10,y=1800)
8605
8606     #Crear botones ventana diseño cortante
8607     boton_aceptar_cortante =
8608     ttk.Button(ventana_cortante,text="Aceptar",command=Crear_grafica_cortante)
8609     boton_aceptar_cortante.place(x=180,y=560,width=100)
8610
8611     boton_agregar_cortante = ttk.Button(ventana_cortante,text="Agregar
datos",command=agregar_datos_treeview_cortante)
8612     boton_agregar_cortante.place(x=360,y=560,width=100)
8613
8614     boton_cancelar_cortante =
8615     ttk.Button(ventana_cortante,text="Cancelar",command=lambda:
cerrar_ventana_v(ventana_cortante))
8616     boton_cancelar_cortante.place(x=540,y=560,width=100)
8617
8618     #Agregar contenido en la pestaña tab_diseño
8619     labelframe_Diseño_V = tk.LabelFrame(tab_diseño,text="")
8620     labelframe_Diseño_V.place(x=10,y=10,width=400,height=40)
8621
8622     # Variables para los Checkbuttons
8623     var_estribo = tk.BooleanVar()
8624     var_grafica = tk.BooleanVar()
8625
8626     # Crear los Checkbuttons dentro del LabelFrame
8627     check_estribo = tk.Checkbutton(labelframe_Diseño_V, text="Estribo de
diseño", variable=var_estribo,command=Estribo_diseño)
8628     check_estribo.place(x=40, y=5)
8629
8630     check_grafica = tk.Checkbutton(labelframe_Diseño_V, text="Mostrar gráfica
de cortante", variable=var_grafica,command=Crear_grafica_cortante)

```

```
8629         check_grafica.place(x=190, y=5)
8630
8631         labelframe_Eschema = tk.LabelFrame(tab_diseño)
8632         labelframe_Eschema.place(x=10,y=60,width=400,height=390)
8633
8634         # Crear el Notebook
8635         Notebook_Eschema =
8636         ttk.Notebook(labelframe_Eschema,style="CustomNotebook.TNotebook") # Crear el
8637         Notebook dentro del LabelFrame
8638
8639         # Crear una pestaña dentro del Notebook
8640         tab_Eschema = ttk.Frame(Notebook_Eschema) # Aquí es donde puedes colocar
8641         el contenido de la pestaña
8642
8643         # Agregar la pestaña al Notebook con el título 'Cálculos previos'
8644         Notebook_Eschema.add(tab_Eschema, text="Cálculos previos")
8645
8646         # Empaquetar el Notebook en el LabelFrame
8647         Notebook_Eschema.pack(fill="both", expand=True)
8648
8649         # Crear un Canvas dentro de la pestaña 'tab_Eschema'
8650         canvas_esquema = tk.Canvas(tab_Eschema) # Ajusta el tamaño según lo
8651         necesario
8652         canvas_esquema.pack(fill="both", expand=True)
8653
8654         # Crear el Scrollbar vertical
8655         scrollbar_esquema = tk.Scrollbar(tab_Eschema,
8656         orient="vertical",command=canvas_esquema.yview)
8657         scrollbar_esquema.place(x=370, y=0, width=20, height=355)
8658
8659         # Vincular el Scrollbar vertical al Canvas
8660         canvas_esquema.config(yscrollcommand=scrollbar_esquema.set)
8661
8662         labelframe_grafica = tk.LabelFrame(canvas_esquema,text="Gráfica de fuerza
8663         cortante [V]")
8664
8665         labelframe_info_1 = tk.LabelFrame(canvas_esquema,text="")
8666
8667         label_info_1 = tk.Label(labelframe_info_1,text="Estribo:")
8668         label_info_1.place(x=10,y=10)
8669
8670         entry_info_estribo = tk.Entry(labelframe_info_1)
8671         entry_info_estribo.place(x=60,y=10,width=100)
8672
8673         label_info_2 = tk.Label(labelframe_info_1,text="Área:")
8674         label_info_2.place(x=180,y=10)
8675
8676         entry_info_area = tk.Entry(labelframe_info_1)
8677         entry_info_area.place(x=220,y=10,width=100)
8678
8679         labelframe_Vu = tk.LabelFrame(canvas_esquema,text="Calcular cortante
8680         último [Vu]")
8681
8682         label_Vu_Wu = tk.Label(labelframe_Vu,text="Wu:")
8683         label_Vu_Wu.place(x=20,y=10)
8684
8685         combo_Vu_Wu = ttk.Combobox(labelframe_Vu)
8686         combo_Vu_Wu.place(x=60,y=10,width=100)
8687         # Asociar la función al evento de selección en el ComboBox
8688         combo_Vu_Wu.bind("<<ComboboxSelected>>", Calcular_Cortante_Vu)
```

```

8682
8683     label_Vu_ultimo = tk.Label(labelframe_Vu,text="Vu:")
8684     label_Vu_ultimo.place(x=180,y=10)
8685
8686     entry_Vu_ultimo = tk.Entry(labelframe_Vu)
8687     entry_Vu_ultimo.place(x=220,y=10,width=100)
8688
8689     labelframe_revision = tk.LabelFrame(canvas_esquema,text="Revisar
condiciones de diseño")
8690
8691     label_φ_revision = tk.Label(labelframe_revision,text="φCortante:")
8692     label_φ_revision.place(x=70,y=10)
8693
8694     combo_φ_revision = ttk.Combobox(labelframe_revision,values=("0.75",))
8695     combo_φ_revision.place(x=150,y=10,width=100)
8696     combo_φ_revision.set("Seleccionar")
8697
8698     label_fc_revision = tk.Label(labelframe_revision,text="Concreto:")
8699     label_fc_revision.place(x=70,y=70)
8700
8701     combo_fc_revision = ttk.Combobox(labelframe_revision,values=("17.5
Mpa", "21 Mpa", "24.5 Mpa", "28 Mpa", "35 Mpa", "42 Mpa"))
8702     combo_fc_revision.place(x=150,y=70,width=100)
8703     combo_fc_revision.set("Seleccionar")
8704     #Asociar la función al evento de selección en el combo_factor_λ
8705     combo_fc_revision.bind("<<ComboboxSelected>>", Calcular_Resistencia_φVc)
8706
8707     label_factor_λ = tk.Label(labelframe_revision,text="Factor λ:")
8708     label_factor_λ.place(x=70,y=40)
8709
8710     combo_factor_λ = ttk.Combobox(labelframe_revision,values=("1.0",))
8711     combo_factor_λ.place(x=150,y=40,width=100)
8712     combo_factor_λ.set("Seleccionar")
8713
8714     label_φVc_revision = tk.Label(labelframe_revision,text="φVc:")
8715     label_φVc_revision.place(x=90,y=100)
8716
8717     entry_φVc_revision = tk.Entry(labelframe_revision)
8718     entry_φVc_revision.place(x=150,y=100,width=100)
8719
8720     label_φVcmedios_revision = tk.Label(labelframe_revision,text="φVc/2:")
8721     label_φVcmedios_revision.place(x=90,y=130)
8722
8723     entry_φVcmedios_revision = tk.Entry(labelframe_revision)
8724     entry_φVcmedios_revision.place(x=150,y=130,width=100)
8725
8726     # Variable que controla el estado del checkbox
8727     check_revisar = tk.BooleanVar()
8728
8729     check_revisar_widget = tk.Checkbutton(canvas_esquema, text="Revisar lo que
resiste el concreto", variable=check_revisar,
command=Revisar_condiciones_diseño)
8730
8731
8732     labelframe_condiciones = tk.LabelFrame(canvas_esquema,text="")
8733
8734     # Crear las variables para los checkboxes
8735     check_condición_1 = tk.BooleanVar()
8736     check_condición_2 = tk.BooleanVar()
8737     check_condición_3 = tk.BooleanVar()
8738

```

```

8739         # Crear los checkboxes
8740         checkbox_1_V = tk.Checkbutton(labelframe_condiciones, text="Sí  $V_u > \phi V_c$ 
[Diseñar refuerzo transversal]", variable=check_condición_1)
8741         checkbox_1_V.place(x=10,y=10)
8742         checkbox_2_V = tk.Checkbutton(labelframe_condiciones, text="Sí  $\phi V_c/2 < V_u < \phi V_c$ 
[Colocar área mínima]", variable=check_condición_2)
8743         checkbox_2_V.place(x=10,y=40)
8744         checkbox_3_V = tk.Checkbutton(labelframe_condiciones, text="Sí  $V_u < \phi V_c/2$ 
[No necesita refuerzo transversal]", variable=check_condición_3)
8745         checkbox_3_V.place(x=10,y=70)
8746
8747         labelframe_mensaje =
tk.LabelFrame(labelframe_condiciones,text="Resistencia a cortante del concreto
[ $\phi V_c$ ]")
8748         labelframe_mensaje.place(x=10,y=100,width=325,height=60)
8749
8750         label_mensaje_condicion =tk.Label(labelframe_mensaje,text="")
8751         label_mensaje_condicion.place(x=10,y=10)
8752
8753         labelframe_resistencia_acero =
tk.LabelFrame(canvas_esquema,text="Resistencia del acero transversal [ $V_s$ ]")
8754
8755         label_resistencia_Vs = tk.Label(labelframe_resistencia_acero,text=" $V_s$ :")
8756         label_resistencia_Vs.place(x=20,y=15)
8757
8758         entry_resistencia_Vs = tk.Entry(labelframe_resistencia_acero)
8759         entry_resistencia_Vs.place(x=60,y=15,width=100)
8760
8761         label_resistencia_phiVc =
tk.Label(labelframe_resistencia_acero,text=" $\phi$ Cortante: 0.75")
8762         label_resistencia_phiVc.place(x=210,y=15)
8763
8764         labelframe_verificacion = tk.LabelFrame(canvas_esquema,text="Control del
tamaño de la sección del concreto")
8765
8766         label_verificacion = tk.Label(labelframe_verificacion,text="Sí  $V_s \leq 66\%$ 
Sección:")
8767         label_verificacion.place(x=220,y=10)
8768
8769         label_verificacion_Vs =
tk.Label(labelframe_verificacion,text="Resistencia:")
8770         label_verificacion_Vs.place(x=10,y=10)
8771
8772         label_copia_Vs =
tk.Label(labelframe_verificacion,text="",bd=2,relief="sunken")
8773         label_copia_Vs.place(x=95,y=10,width=80)
8774
8775         label_verificacion_seccion = tk.Label(labelframe_verificacion,text="66%
Sección:")
8776         label_verificacion_seccion.place(x=10,y=35)
8777
8778         label_copia_seccion =
tk.Label(labelframe_verificacion,text="",bd=2,relief="sunken")
8779         label_copia_seccion.place(x=95,y=35,width=80)
8780
8781         label_cumple_Vs =
tk.Label(labelframe_verificacion,text="",bd=2,relief="sunken")
8782         label_cumple_Vs.place(x=240,y=35,width=80)
8783
8784         label_conclusion_V = tk.Label(tab_diseño,text="",

```

```

8785         bd=2,
8786         relief="sunken",
8787         justify="left",
8788         anchor="center",
8789         wraplength=380)
8790     label_conclusion_V.place(x=10,y=450,width=400,height=57)
8791
8792     labelframe_separacion_req = tk.LabelFrame(tab_diseño,text="Separación
máxima según la NSR-10 [Vs]")
8793     labelframe_separacion_req.place(x=420,y=2,width=395,height=60)
8794
8795     # Crear un IntVar para controlar el estado del Checkbutton
8796     var_S = tk.IntVar(value=0)
8797
8798     radiobutton_S = tk.Radiobutton(labelframe_separacion_req,text="Calcular
Smax",variable=var_S,command=Calcular_Smax)
8799     radiobutton_S.place(x=10,y=5)
8800
8801     label_Smax= tk.Label(labelframe_separacion_req,text="Smax:")
8802     label_Smax.place(x=200,y=5)
8803
8804     # Crear una variable de tipo StringVar para monitorear el cambio en
entry_Smax
8805     smax_var = tk.StringVar()
8806
8807     # Asociar el StringVar con el entry_Smax
8808     entry_Smax= tk.Entry(labelframe_separacion_req)
8809     entry_Smax.place(x=250,y=5,width=100)
8810     entry_Smax.config(textvariable=smax_var)
8811
8812     # Usar trace para que llame a la función cada vez que el valor de
entry_Smax cambie
8813     smax_var.trace("w", actualizar_label_Smax)
8814
8815     labelframe_Av = tk.LabelFrame(tab_diseño,text="Calcular el acero de
refuerzo transversal")
8816     labelframe_Av.place(x=420,y=63,width=395,height=60)
8817
8818     # Crear un IntVar para controlar el estado del Checkbutton
8819     var_Av = tk.IntVar(value=0)
8820
8821     radiobutton_Av = tk.Radiobutton(labelframe_Av,text="Calcular
Av",variable=var_Av,command=Calcular_Av)
8822     radiobutton_Av.place(x=10,y=5)
8823
8824     label_Av = tk.Label(labelframe_Av,text="Av:")
8825     label_Av.place(x=205,y=5)
8826
8827     entry_Av = tk.Entry(labelframe_Av)
8828     entry_Av.place(x=250,y=5,width=100)
8829
8830     labelframe_calcular_ramas = tk.LabelFrame(tab_diseño,text="Calcular el
número de ramas de estribos")
8831     labelframe_calcular_ramas.place(x=420,y=125,width=395,height=60)
8832
8833     # Crear un IntVar para controlar el estado del Checkbutton
8834     var_ramas = tk.IntVar(value=0)
8835
8836     radiobutton_ramas =
tk.Radiobutton(labelframe_calcular_ramas,text="Calcular

```

```

#ramas",variable=var_ramas,command=Calcular_numero_ramas)
8837     radiobutton_ramas.place(x=10,y=5)
8838
8839     label_ramas = tk.Label(labelframe_calcular_ramas,text="#ramas:")
8840     label_ramas.place(x=190,y=5)
8841
8842     entry_ramas = tk.Entry(labelframe_calcular_ramas)
8843     entry_ramas.place(x=250,y=5,width=100)
8844
8845     label_ramas_V = tk.Label(tab_diseño,text="",bd=2,relief="sunken",
8846                             justify="left",
8847                             anchor="center",
8848                             wraplength=380)
8849     label_ramas_V.place(x=420,y=190,width=395,height=50)
8850
8851     labelframe_resistencia_estribos = tk.LabelFrame(tab_diseño,text="Calcular
resistencia de los estribos en función de la separación")
8852     labelframe_resistencia_estribos.place(x=420,y=245,width=395,height=60)
8853
8854     var_resistencia = tk.IntVar(value=0)
8855     radiobutton_resistencia =
tk.Radiobutton(labelframe_resistencia_estribos,text="Calcular
resistencia",variable=var_resistencia,
8856                 command=Calcular_Vs_resistencia_estribos)
8857     radiobutton_resistencia.place(x=10,y=5)
8858
8859     label_Vs_estribos = tk.Label(labelframe_resistencia_estribos,text="Vs:")
8860     label_Vs_estribos.place(x=200,y=5)
8861
8862     entry_Vs_estribos = tk.Entry(labelframe_resistencia_estribos)
8863     entry_Vs_estribos.place(x=250,y=5,width=100)
8864
8865     label_Vs_estribos_1 = tk.Label(tab_diseño,text="",bd=2,relief="sunken",
8866                                   justify="left",
8867                                   anchor="center",
8868                                   wraplength=380)
8869     label_Vs_estribos_1.place(x=420,y=310,width=395,height=50)
8870
8871     labelframe_disposicion_estribos =
tk.LabelFrame(tab_diseño,text="Disposición de los estribos")
8872     labelframe_disposicion_estribos.place(x=420,y=365,width=395,height=140)
8873
8874     label_disposicion_estribos =
tk.Label(labelframe_disposicion_estribos,text="Los estribos deben ser colocados de
manera que se extiendan,como mínimo, hasta la sección en la que el esfuerzo
cortante alcance el valor de  $V_u=1/2 \phi V_c$ .\n"
8875         "Se recomienda colocar el primer
estribo cerca de la cara del apoyo, a unos 5cm, ya que en esa zona la magnitud
del cortante tambien es alta.Además,esta practica es conveniente en las obras, ya
que facilita la colocación de los estribos.",
8876         justify="left",
8877         anchor="center",
8878         wraplength=380)
8879     label_disposicion_estribos.place(x=10,y=1)
8880
8881     labelframe_optimizacion = tk.LabelFrame(tad_optimización,text="")
8882     labelframe_optimizacion.place(x=10,y=10,width=400,height=60)
8883
8884     # Variable que controla el estado del checkbox
8885     check_resistencia = tk.BooleanVar()

```

```

8886     check_resistencia_total = tk.BooleanVar()
8887     check_resistencia = tk.Checkbutton(labelframe_optimizacion, text="Calcular
resistencia requerida [Vs]", variable=check_resistencia,
8888                                     command=Calcular_resistencia_Vs_requerida)
8889     check_resistencia.place(x=10, y=5)
8890     check_resistencia_total = tk.Checkbutton(labelframe_optimizacion,
text="Calcular resistencia total [ $\phi V_n$ ]", variable=check_resistencia_total,
8891                                     command=Calcular_resistencia_total_Vn)
8892     check_resistencia_total.place(x=10, y=30)
8893
8894     entry_resistencia_Vs_requerida = tk.Entry(labelframe_optimizacion)
8895     #entry_resistencia_Vs_requerida.place(x=220,y=5,width=100)
8896
8897     # Crear la tabla Treeview en la pestaña 'tad_optimización'
8898     treeview_cortante = ttk.Treeview(tad_optimización, columns=("Vs", "Vu",
" $\phi V_n$ ", " $Vu \leq \phi V_n$ "), show="headings")
8899     treeview_cortante.place(x=10, y=80, width=400, height=80)
8900
8901     # Definir las columnas
8902     treeview_cortante.heading("Vs", text="Vs")
8903     treeview_cortante.heading("Vu", text="Vu")
8904     treeview_cortante.heading(" $\phi V_n$ ", text=" $\phi V_n$ ")
8905     treeview_cortante.heading(" $Vu \leq \phi V_n$ ", text=" $Vu \leq \phi V_n$ ")
8906
8907     # Configurar el ancho de las columnas
8908     treeview_cortante.column("Vs", width=50, anchor="center")
8909     treeview_cortante.column("Vu", width=50, anchor="center")
8910     treeview_cortante.column(" $\phi V_n$ ", width=50, anchor="center")
8911     treeview_cortante.column(" $Vu \leq \phi V_n$ ", width=50, anchor="center")
8912
8913
8914     def no_resize(event):
8915         # Restaurar el tamaño original de las columnas usando un ciclo
8916         for col in treeview_cortante["columns"]:
8917             treeview_cortante.column(col, width=99, anchor="center")
8918
8919     # Vincular el evento de clic con la función no_resize
8920     treeview_cortante.bind("<ButtonRelease-1>", no_resize)
8921
8922     label_resistencia_total =
tk.Label(tad_optimización, text="", bd=2, relief="sunken",
8923         justify="left",
8924         anchor="center",
8925         wraplength=380)
8926     label_resistencia_total.place(x=10, y=170, width=400, height=50)
8927
8928     check_valores = tk.BooleanVar()
8929     check_valores = tk.Checkbutton(tad_optimización, text="Valores críticos y
de diseño cortante", variable=check_valores,
8930                                     command=Insertar_datos_cortante_diseño)
8931
8932     check_valores.place(x=10, y=220)
8933
8934     # Crear la tabla Treeview en la pestaña 'tad_optimización'
8935     treeview_cortante_valores = ttk.Treeview(tad_optimización, columns=
("Vmax", "Vu", " $\phi V_c$ ", " $\phi V_c/2$ "), show="headings")
8936     treeview_cortante_valores.place(x=10, y=250, width=400, height=80)
8937
8938     # Definir las columnas
8939     treeview_cortante_valores.heading("Vmax", text="Vmax")

```

```

8940 treeview_cortante_valores.heading("Vu", text="Vu")
8941 treeview_cortante_valores.heading("φVc", text="φVc")
8942 treeview_cortante_valores.heading("φVc/2", text="φVc/2")
8943
8944 # Configurar el ancho de las columnas
8945 treeview_cortante_valores.column("Vmax", width=50,anchor="center")
8946 treeview_cortante_valores.column("Vu", width=50,anchor="center")
8947 treeview_cortante_valores.column("φVc", width=50,anchor="center")
8948 treeview_cortante_valores.column("φVc/2", width=50,anchor="center")
8949
8950
8951 def no_resize(event):
8952     # Restaurar el tamaño original de las columnas usando un ciclo
8953     for col in treeview_cortante_valores["columns"]:
8954         treeview_cortante_valores.column(col, width=99, anchor="center")
8955
8956     # Vincular el evento de clic con la función no_resize
8957     treeview_cortante_valores.bind("<ButtonRelease-1>", no_resize)
8958
8959     labelframe_columna = tk.LabelFrame(tad_optimización,text="Ancho mínimo de
la columna")
8960     labelframe_columna.place(x=10,y=340,width=400,height=150)
8961
8962     label_ancho_C = tk.Label(labelframe_columna,text="Ancho columna:")
8963     label_ancho_C.place(x=10,y=20)
8964
8965     entry_ancho_C = tk.Entry(labelframe_columna)
8966     entry_ancho_C.place(x=120,y=20,width=100)
8967
8968     label_borde_C = tk.Label(labelframe_columna,text="Borde columna:")
8969     label_borde_C.place(x=10,y=50)
8970
8971     entry_borde_C = tk.Entry(labelframe_columna)
8972     entry_borde_C.place(x=120,y=50,width=100)
8973
8974     label_mensaje_C = tk.Label(labelframe_columna,text="El ancho mínimo de la
columna fue determinado a partir del análisis de la longitud de desarrollo de
anclaje realizado durante el diseño a flexión",
8975                               justify="left",
8976                               anchor="center",
8977                               wraplength=380)
8978     label_mensaje_C.place(x=10,y=80,width=380)
8979
8980     check_longitud_refuerzo_var = tk.BooleanVar()
8981     check_longitud_refuerzo = tk.Checkbutton(tad_optimización,
text="Longitudes de refuerzo cortante", variable=check_longitud_refuerzo_var,
8982     command=dibujar_longitud_triangulo_cortante)
8983
8984     check_longitud_refuerzo.place(x=420,y=10)
8985
8986     frame_longitud_refuerzo = tk.Frame(tad_optimización,bd=2,relief="ridge")
8987     frame_longitud_refuerzo.place(x=420,y=40,width=395,height=250)
8988
8989     # Crear el Notebook
8990     Notebook_longitud =
ttk.Notebook(frame_longitud_refuerzo,style="CustomNotebook.TNotebook") # Crear el
Notebook dentro del LabelFrame
8991
8992     # Crear una pestaña dentro del Notebook

```

```
8993         tab_longitud = ttk.Frame(Notebook_longitud) # Aquí es donde puedes
colocar el contenido de la pestaña
8994
8995         # Agregar la pestaña al Notebook
8996         Notebook_longitud.add(tab_longitud, text="Longitud triángulo")
8997
8998         # Empaquetar el Notebook en el LabelFrame
8999         Notebook_longitud.pack(fill="both", expand=True)
9000
9001         # Crear un Canvas dentro de la pestaña 'tab_longitud'
9002         canvas_longitud = tk.Canvas(tab_longitud,bg="white")
9003         canvas_longitud.pack(fill="both", expand=True)
9004
9005         # Crear frame_estribos_longitudes
9006
9007         frame_estribos_longitudes = tk.Frame(tab_optimización,bd=2,relief="ridge")
9008         frame_estribos_longitudes.place(x=420,y=290,width=395,height=200)
9009
9010         # Crear el Notebook
9011         Notebook_estribos =
ttk.Notebook(frame_estribos_longitudes,style="CustomNotebook.TNotebook")
9012
9013         # Crear una pestaña dentro del Notebook
9014         tab_estribos = ttk.Frame(Notebook_estribos)
9015
9016         # Agregar la pestaña al Notebook
9017         Notebook_estribos.add(tab_estribos, text="Ubicación refuerzo transversal")
9018
9019         # Empaquetar el Notebook en el LabelFrame
9020         Notebook_estribos.pack(fill="both", expand=True)
9021
9022         # Crear un Canvas dentro de la pestaña 'tab_estribos'
9023         canvas_estribos = tk.Canvas(tab_estribos)
9024         canvas_estribos.pack(fill="both", expand=True)
9025
9026         # Crear el Scrollbar vertical
9027         scrollbar_estribos = tk.Scrollbar(tab_estribos,
orient="vertical",command=canvas_estribos.yview)
9028         scrollbar_estribos.place(x=370, y=0, width=20, height=170)
9029
9030         # Vincular el Scrollbar vertical al Canvas
9031         canvas_estribos.config(yscrollcommand=scrollbar_estribos.set)
9032
9033         # Crear labelframe_estribos
9034         labelframe_estribos = tk.LabelFrame(tab_estribos)
9035
9036         # Crear Slider
9037         slider_cortante = tk.Scale(labelframe_estribos, from_=0.00,
orient="horizontal", resolution=0.00)
9038         slider_cortante.place(x=250,y=1)
9039
9040         label_posicion = tk.Label(labelframe_estribos,text="Posición [L]:")
9041         label_posicion.place(x=10,y=10)
9042
9043         entry_posicion = tk.Entry(labelframe_estribos)
9044         entry_posicion.place(x=90,y=10,width=80)
9045
9046         # Crear el label para mostrar el valor del slider
9047         label_valor = tk.Label(labelframe_estribos, text="",bd=2,relief="ridge")
9048         label_valor.place(x=253,y=1,width=100)
```

```

9049
9050     labelframe_l_refuerzo = tk.LabelFrame(tab_estribos,text="")
9051
9052     # Función para redimensionar la imagen
9053     def redimensionar_imagen(ruta_imagen, tamano):
9054         imagen_agregar_eliminar = Image.open(ruta_imagen)
9055         imagen_agregar_eliminar = imagen_agregar_eliminar.resize(tamano)
9056         imagen_longitud_refuerzo = Image.open(ruta_imagen)
9057         imagen_longitud_refuerzo = imagen_longitud_refuerzo.resize(tamano)
9058         return
ImageTk.PhotoImage(imagen_agregar_eliminar,imagen_longitud_refuerzo)
9059
9060     # Cargar las imágenes de los iconos redimensionadas
9061     imagen_agregar =
redimensionar_imagen("Iconos_proyecto/agregar_marcador.png", (20, 20))
9062     imagen_eliminar =
redimensionar_imagen("Iconos_proyecto/eliminar_marcador.png", (20, 20))
9063     imagen_refuerzo = redimensionar_imagen("Iconos_proyecto/mano_de_obra.png",
(30, 30))
9064
9065     boton_agregar_marcador =
tk.Button(labelframe_estribos,image=imagen_agregar,bd=2,relief="flat")
9066     boton_agregar_marcador.place(x=180,y=10)
9067
9068     boton_eliminar_marcador =
tk.Button(labelframe_estribos,image=imagen_eliminar,bd=2,relief="flat")
9069     boton_eliminar_marcador.place(x=215,y=10)
9070
9071     # Mantener una referencia de las imágenes
9072     boton_agregar_marcador.icono = imagen_agregar
9073     boton_eliminar_marcador.icono = imagen_eliminar
9074
9075     frame_distribucion_estribo =
tk.Frame(labelframe_l_refuerzo,bd=2,relief="sunken")
9076     frame_distribucion_estribo.place(x=10,y=10,width=330,height=80)
9077
9078     check_primer_refuerzo_var = tk.BooleanVar()
9079     check_longitud_estribo = tk.Checkbutton(frame_distribucion_estribo,
text="Disposición del primer estribo a 5 cm", variable=check_primer_refuerzo_var,
command=actualizar_dato_primer_estribo)
9080     check_longitud_estribo.place(x=10,y=10)
9081
9082
9083     label_primer_estribo =
tk.Label(frame_distribucion_estribo,text="",bd=2,relief="sunken")
9084     label_primer_estribo.place(x=240,y=10,width=80)
9085
9086     boton_longitud_refuerzo =
tk.Button(tad_optimización,image=imagen_refuerzo,bd=2,relief="flat",command=abrir_
subventana_refuerzo)
9087     boton_longitud_refuerzo.place(x=780,y=4)
9088
9089     # Mantener una referencia de las imágenes
9090     boton_longitud_refuerzo.icono = imagen_refuerzo
9091
9092     # Crear el Scrollbar vertical
9093     scrollbar_nsr10 = tk.Scrollbar(tab_requisitos_NSR10,
orient="vertical",command=canvas_nsr10.yview)
9094     scrollbar_nsr10.place(x=360, y=0, width=20, height=370)
9095
9096     # Vincular el Scrollbar vertical al Canvas

```

```
9097         canvas_nsr10.config(yscrollcommand=scrollbar_nsr10.set)
9098
9099         # Crear el Scrollbar vertical
9100         scrollbar_cortante = tk.Scrollbar(tab_generalidades, orient="vertical",
command=canvas_cortante.yview)
9101         scrollbar_cortante.place(x=370, y=0, width=20, height=460)
9102
9103         # Vincular el Scrollbar vertical al Canvas
9104         canvas_cortante.config(yscrollcommand=scrollbar_cortante.set)
9105
9106         canvas_cortante.create_window((185,70), window=labelframe_V, width=345,
height=100)
9107         canvas_cortante.create_window((185,190),window =frame_V, width=300,
height=100)
9108         canvas_cortante.create_window((185,310),window =labelframe_V_info,
width=345, height=100)
9109         canvas_cortante.create_window((185,430),window =labelframe_V_falla,
width=345, height=100)
9110         canvas_cortante.create_window((185,555),window =frame_V_falla, width=250,
height=120)
9111         canvas_cortante.create_window((185,760),window =labelframe_V_1, width=345,
height=240)
9112         canvas_nsr10.create_window((180,40),window=labelframe_req_1, width=335,
height=60)
9113         canvas_nsr10.create_window((180,1065),window=labelframe_req_2, width=335,
height=1960)
9114         canvas_esquema.create_window((185,160), window=labelframe_grafica,
width=350, height=300)
9115         canvas_esquema.create_window((185,350),window=labelframe_info_1,
width=350, height=50)
9116         canvas_esquema.create_window((185,420),window=labelframe_Vu, width=350,
height=70)
9117         canvas_esquema.create_window((185,555),window=labelframe_revision,
width=350, height=180)
9118         canvas_esquema.create_window((110,660),window=check_revisar_widget)
9119         canvas_esquema.create_window((185,770),window=labelframe_condiciones,
width=350, height=180)
9120
9121         canvas_esquema.create_window((185,910),window=labelframe_resistencia_acero,
width=350, height=80)
9122         canvas_esquema.create_window((185,1000),window=labelframe_verificacion,
width=350, height=80)
9123         canvas_estribos.create_window((185,30),window=labelframe_estribos,
width=360, height=50)
9124         canvas_estribos.create_window((185,150),window=labelframe_l_refuerzo,
width=360, height=180)
9125
9126         # Función para actualizar el área de desplazamiento del canvas
9127         def update_scrollregion(event=None):
9128             bbox = canvas_estribos.bbox("all") # Obtener las coordenadas de todo
el contenido
9129             if bbox:
9130                 # Ajustar el área de desplazamiento añadiendo márgenes
9131                 canvas_estribos.config(scrollregion=(0, 0, bbox[2] + 50, bbox[3] +
10))
9132
9133         # Llamar a update_scrollregion para ajustar el área de desplazamiento
9134         canvas_estribos.bind("<Configure>", update_scrollregion)
9135
9136         # Función para actualizar el área de desplazamiento del canvas
```

```
9136     def update_scrollregion(event=None):
9137         bbox = canvas_cortante.bbox("all") # Obtener las coordenadas de todo
el contenido
9138         if bbox:
9139             # Ajustar el área de desplazamiento añadiendo márgenes
9140             canvas_cortante.config(scrollregion=(0, 0, bbox[2] + 50, bbox[3] +
10))
9141
9142             # Llamar a update_scrollregion para ajustar el área de desplazamiento
9143             canvas_cortante.bind("<Configure>", update_scrollregion)
9144
9145             # Función para actualizar el área de desplazamiento del canvas
9146             def update_scrollregion(event=None):
9147                 bbox = canvas_nsr10.bbox("all") # Obtener las coordenadas de todo el
contenido
9148                 if bbox:
9149                     # Ajustar el área de desplazamiento añadiendo márgenes
9150                     canvas_nsr10.config(scrollregion=(0, 0, bbox[2] + 50, bbox[3] +
10))
9151
9152                     # Llamar a update_scrollregion para ajustar el área de desplazamiento
9153                     canvas_nsr10.bind("<Configure>", update_scrollregion)
9154
9155                     # Función para actualizar el área de desplazamiento del canvas
9156                     def update_scrollregion(event=None):
9157                         bbox = canvas_esquema.bbox("all") # Obtener las coordenadas de todo
el contenido
9158                         if bbox:
9159                             # Ajustar el área de desplazamiento añadiendo márgenes
9160                             canvas_esquema.config(scrollregion=(0, 0, bbox[2] + 50, bbox[3] +
10))
9161
9162                             # Llamar a update_scrollregion para ajustar el área de desplazamiento
9163                             canvas_esquema.bind("<Configure>", update_scrollregion)
9164
9165                             # Marcar que la ventana ahora está abierta
9166                             ventana_abierta = True
9167
9168                             # Cuando la ventana se cierre, ejecutar la función para marcar que está
cerrada
9169                             ventana_cortante.protocol("WM_DELETE_WINDOW", lambda:
cerrar_ventana_v(ventana_cortante))
9170
9171
9172     def cerrar_ventana_v(ventana_cortante):
9173         global ventana_abierta
9174         ventana_abierta = False
9175         ventana_cortante.destroy()
9176
9177     def actualizar_dato_primer_estribo():
9178         # Obtener el texto del Entry
9179         texto_entry = entry_borde_C.get()
9180
9181         # Extraer solo los números del texto
9182         numeros = re.findall(r'\d+\.?\d*', texto_entry)
9183         valor_numerico_1 = float(numeros[0]) if numeros else 0.0 # Convertir a float
9184
9185         # Calcular el nuevo valor
9186         Primer_estribo = valor_numerico_1 + 0.05
9187
```

```

9188 # Actualizar el Label con el nuevo valor
9189 label_primer_estribo.config(text=f"{Primer_estribo}m")
9190
9191
9192 def Crear_grafica_cortante():
9193     # Verificar si el gráfico ya existe y eliminarlo si es necesario
9194     if hasattr(Crear_grafica_cortante, 'canvas'):
9195         Crear_grafica_cortante.canvas.get_tk_widget().destroy()
9196
9197     # Obtener los valores de las variables
9198     try:
9199         Wu_cortante = float(label_seccion_Wu.cget("text")) # Carga distribuida
9200         L_cortante = float(label_seccion_luz.cget("text")) # Longitud de la viga
9201     except ValueError:
9202         print("Error: Los valores de Wu_cortante y L_cortante deben ser
9203             numéricos.")
9204         return
9205
9206     # Actualizar el valor de Wu_cortante en el combo (agregarlo como nuevo ítem en
9207     la lista)
9208     nuevo_valor = f"{Wu_cortante:.2f} kN/m"
9209
9210     # Verificar si el valor ya está en el combobox, si no, agregarlo
9211     if nuevo_valor not in combo_Vu_Wu['values']:
9212         combo_Vu_Wu['values'] = (*combo_Vu_Wu['values'], nuevo_valor)
9213
9214     # Dejar "Seleccionar" como el valor visible para el usuario
9215     combo_Vu_Wu.set("Seleccionar")
9216
9217     # Calcular reacciones en los apoyos
9218     Reaccion_cortante_apoyos = (Wu_cortante * L_cortante) / 2
9219     Ray_cortante = Reaccion_cortante_apoyos
9220     Rby_cortante = Ray_cortante # Viga simétrica
9221
9222     # Actualizar el label con el valor de la Reacción Cortante en los apoyos
9223     label_Vmax_prueba.config(text=f"{Reaccion_cortante_apoyos:.2f} kN")
9224
9225     # Crear figura de matplotlib
9226     fig = plt.Figure(figsize=(10, 10), dpi=70)
9227     ax = fig.add_subplot(111)
9228
9229     # Calcular puntos clave del diagrama
9230     x = [0, L_cortante / 2, L_cortante]
9231     y = [Ray_cortante, 0, -Rby_cortante] # Valores de la fuerza cortante en los
9232     puntos clave
9233
9234     # Graficar la fuerza cortante
9235     ax.plot(x, y, label='Fuerza Cortante', color='blue', linewidth=2)
9236
9237     # Sombrear el área positiva
9238     x_positiva = [0, L_cortante / 2]
9239     y_positiva = [Ray_cortante, 0]
9240     ax.fill_between(x_positiva, y_positiva, 0, color='Darkblue', alpha=1)
9241
9242     # Sombrear el área negativa
9243     x_negativa = [L_cortante / 2, L_cortante]
9244     y_negativa = [0, -Rby_cortante]
9245     ax.fill_between(x_negativa, y_negativa, 0, color='red', alpha=1)

```

```

9243
9244 # Agregar marcadores en Ray_cortante y Rby_cortante
9245 ax.scatter(0, Ray_cortante, color='green', label=f'Vmax: {Ray_cortante:.2f}
kN', s=50, zorder=5)
9246 ax.scatter(L_cortante, -Rby_cortante, color='green', s=50, zorder=5)
9247
9248 # Agregar leyenda para L_cortante (línea sin marcador)
9249 ax.plot([], [], color='black', label=f'Luz: {L_cortante:.2f} m', linewidth=2)
9250
9251 # Configurar ejes y leyendas
9252 ax.axhline(0, color='black', linewidth=1) # Línea horizontal en y=0
9253 ax.set_xlim(0, L_cortante)
9254 ax.set_ylim(min(y) - 10, max(y) + 10)
9255
9256 # Personalizar texto de los ejes en negrita
9257 ax.set_xlabel("Longitud de la viga (m)", fontweight="bold")
9258 ax.set_ylabel("Fuerza Cortante (kN)", fontweight="bold")
9259 ax.yaxis.set_label_position("right")
9260
9261 # Configurar la leyenda con borde negro
9262 legend = ax.legend(frameon=True)
9263 legend.get_frame().set_edgecolor("black") # Establecer borde negro
9264 legend.get_frame().set_linewidth(1.5) # Grosor del borde
9265
9266 ax.grid(True)
9267
9268 # Dibujar la gráfica en la interfaz
9269 canvas = FigureCanvasTkAgg(fig, master=labelframe_grafica)
9270 canvas.draw()
9271 canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
9272
9273 # Guardar la imagen con alta resolución en la ruta especificada
9274 ruta_imagen = os.path.join("Reportes", "imagenes_pilot",
"grafica_cortante.png")
9275 if not os.path.exists(os.path.dirname(ruta_imagen)):
9276     os.makedirs(os.path.dirname(ruta_imagen))
9277
9278 fig.savefig(ruta_imagen, dpi=300)
9279
9280 # Guardar referencia al canvas
9281 Crear_grafica_cortante.canvas = canvas
9282
9283 # Función que se ejecutará cuando se active o desactive el Checkbutton
9284 def Estribo_diseño():
9285     # Recuperar el valor que tiene la label_prueba_estribo
9286     valor = label_prueba_estribo.cget("text")
9287
9288     if valor == "9.5 mm":
9289         entry_info_estribo.delete(0, tk.END) # Limpiar cualquier texto anterior
9290         entry_info_estribo.insert(0, "E#3") # Actualizar con "E#3"
9291         entry_info_area.delete(0, tk.END) # Limpiar cualquier valor anterior
9292         entry_info_area.insert(0, "71 mm²") # Actualizar área con "71 mm²"
9293
9294     elif valor == "12.7 mm":
9295         entry_info_estribo.delete(0, tk.END) # Limpiar cualquier texto anterior
9296         entry_info_estribo.insert(0, "E#4") # Actualizar con "E#4"
9297         entry_info_area.delete(0, tk.END) # Limpiar cualquier valor anterior
9298         entry_info_area.insert(0, "129 mm²") # Actualizar área con "129 mm²"
9299
9300

```

```

9301 def Calcular_Cortante_Vu(event=None):
9302     # Obtener los valores de los labels y el combobox
9303     try:
9304         # Obtener el valor de Vmax_V (solo el número sin "kN")
9305         Vmax_V = float(label_Vmax_prueba.cget("text").split()[0])
9306         #print(f"Valor de Vmax_V: {Vmax_V} kN")
9307
9308         # Obtener el valor de Wu_V (solo el número sin "kN/m")
9309         Wu_V = float(combo_Vu_Wu.get().split()[0])
9310         #print(f"Valor de Wu_V: {Wu_V} kN/m")
9311
9312         # Obtener el valor de d_V desde el label
9313         d_V = float(label_profundidad_prueba.cget("text").split()[0])
9314         #print(f"Valor de d_V: {d_V} m")
9315
9316         # Obtener el valor de base_Vu_V desde el label (label_ancho_columna)
9317         base_Vu_V = float(label_ancho_columna.cget("text").split()[0]) / 100
9318         #print(f"Valor de base_Vu_V: {base_Vu_V} m")
9319
9320     except ValueError:
9321         print("Error: Los valores deben ser numéricos.")
9322         return
9323
9324     # Calcular Vu_V con la nueva ecuación
9325     Vu_V = Vmax_V - Wu_V * (0.5 * base_Vu_V + d_V)
9326     #print(f"Valor calculado de Vu_V: {Vu_V} kN")
9327
9328     # Mostrar el resultado en el Entry
9329     #print("Procediendo a borrar entry_Vu_ultimo...")
9330     entry_Vu_ultimo.delete(0, tk.END)
9331     #print("Procediendo a insertar el nuevo Vu_V en entry_Vu_ultimo...")
9332     entry_Vu_ultimo.insert(0, f"{Vu_V:.2f} kN")
9333
9334     # Actualizar el valor en el Label
9335     label_Vu.config(text=f"{Vu_V:.2f} kN")
9336
9337 def Calcular_Resistencia_φVc(event=None):
9338     try:
9339         # Obtener el valor de φ_resistencia desde el combo
9340         φ_resistencia = combo_φ_revision.get()
9341         if φ_resistencia == "Seleccionar":
9342             raise ValueError("Por favor, seleccione un valor para φ_resistencia.")
9343         φ_resistencia = float(φ_resistencia)
9344
9345         # Obtener el valor de λ_resistencia desde el combo
9346         λ_resistencia = combo_factor_λ.get()
9347         if λ_resistencia == "Seleccionar":
9348             raise ValueError("Por favor, seleccione un valor para λ_resistencia.")
9349         λ_resistencia = float(λ_resistencia)
9350
9351         # Obtener el valor de f'c_resistencia desde el combo
9352         fc_text = combo_fc_revision.get()
9353         if fc_text == "Seleccionar":
9354             raise ValueError("Por favor, seleccione un valor para
9355 f'c_resistencia.")
9356
9357         match = re.match(r"([0-9.]+)\s*(Mpa|MPa)", fc_text)
9358         if match:
9359             f_c_resistencia = float(match.group(1))

```

```

9360         else:
9361             raise ValueError("f'c_resistencia no tiene el formato esperado.
Asegúrese de seleccionar un valor con 'Mpa' o 'MPA'.")
9362
9363         # Obtener los valores de los labels (b_resistencia y d_resistencia)
9364         b_resistencia = label_b_prueba.cget("text").strip()
9365         if not b_resistencia.replace('.', '', 1).isdigit():
9366             raise ValueError("El valor de b_resistencia no es válido.")
9367         b_resistencia = float(b_resistencia)
9368
9369         d_resistencia = label_profundidad_prueba.cget("text").strip()
9370         if not d_resistencia.replace('.', '', 1).isdigit():
9371             raise ValueError("El valor de d_resistencia no es válido.")
9372         d_resistencia = float(d_resistencia)
9373
9374     except ValueError as e:
9375         print(f"Error: {e}")
9376         return
9377
9378     # Calcular la resistencia al cortante  $\phi V_c$ 
9379      $\phi V_c$ _resistencia =  $\phi$ _resistencia * 0.17 *  $\lambda$ _resistencia *
math.sqrt(f_c_resistencia) * b_resistencia * d_resistencia * 1000
9380
9381     # Mostrar el resultado en el Entry
9382     entry_ $\phi V_c$ _revision.delete(0, tk.END)
9383     entry_ $\phi V_c$ _revision.insert(0, f"{ $\phi V_c$ _resistencia:.2f} kN")
9384
9385     # Actualizar el label_ $\phi V_c$  con el valor calculado
9386     label_ $\phi V_c$ .config(text=f"{ $\phi V_c$ _resistencia:.2f} kN") # Actualiza el label
9387
9388     # Calcular la mitad de la resistencia al cortante  $\phi V_c/2$ 
9389      $\phi V_c$ _media_resistencia =  $\phi V_c$ _resistencia / 2
9390
9391     # Mostrar el resultado en el Entry de la mitad
9392     entry_ $\phi V_c$ medios_revision.delete(0, tk.END)
9393     entry_ $\phi V_c$ medios_revision.insert(0, f"{ $\phi V_c$ _media_resistencia:.2f} kN")
9394
9395     # Actualizar el label_ $\phi V_c/2$  con la mitad de la resistencia calculada
9396     label_ $\phi V_c/2$ .config(text=f"{ $\phi V_c$ _media_resistencia:.2f} kN") # Actualiza el
label
9397
9398     # Obtener el valor de  $V_u$  estribo desde entry_ $V_u$  ultimo
9399      $V_u$ _estribo_text = entry_ $V_u$  ultimo.get().strip()
9400     if  $V_u$ _estribo_text.endswith("kN"):
9401          $V_u$ _estribo_value = float( $V_u$ _estribo_text.replace("kN", "").strip())
9402     else:
9403         raise ValueError("El valor de  $V_u$  estribo no tiene el formato esperado.")
9404
9405     # Calcular la resistencia de los estribos ( $V_s$  estribo)
9406      $V_s$ _estribo = ( $V_u$ _estribo_value -  $\phi V_c$ _resistencia) / 0.75
9407
9408     # Mostrar el resultado en el Entry de resistencia de los estribos
9409     entry_resistencia_ $V_s$ .delete(0, tk.END)
9410     entry_resistencia_ $V_s$ .insert(0, f"{ $V_s$ _estribo:.2f} kN")
9411
9412     # Actualizar el label_copia_ $V_s$  con el resultado de  $V_s$  estribo
9413     label_copia_ $V_s$ .config(text=f"{ $V_s$ _estribo:.2f} kN")
9414
9415     # Actualizar el label  $V_s$  resistencia con el resultado de  $V_s$  estribo
9416      $V_s$ _resistencia.config(text=f"{ $V_s$ _estribo:.2f} kN")

```

```

9417
9418     # Calcular el 66% de la sección
9419     seccion_66 = 0.66 * math.sqrt(f_c_resistencia) * b_resistencia * d_resistencia
* 1000
9420
9421     # Mostrar el resultado en el label_copia_seccion
9422     label_copia_seccion.config(text=f"{seccion_66:.2f} kN")
9423
9424     # Comparar Vs_estribo con el 66% de la sección y actualizar el label_cumple_Vs
9425     if Vs_estribo < seccion_66:
9426         label_cumple_Vs.config(text="Si Cumple")
9427         # Actualizar el label_conclusion_V con el texto de "Cumple"
9428         label_conclusion_V.config(text="Con esto, se concluye que el 66% de la
resistencia de la sección de concreto cumple con la condición de que la sección
transversal diseñada es adecuada.")
9429         # Actualizar label_check_control
9430         label_check_control.config(text="Cumple: Se concluye que el 66% de la
resistencia de la sección de concreto cumple con la condición de que la sección
transversal diseñada es adecuada.")
9431         label_icon_cortante_1.config(image=icon_check_on)
9432     else:
9433         label_cumple_Vs.config(text="No Cumple")
9434         # Actualizar el label_conclusion_V con el texto de "No Cumple"
9435         label_conclusion_V.config(text="En este caso, se concluye que el 66% de la
resistencia de la sección de concreto no cumple con el requisito de que la sección
transversal diseñada sea adecuada.")
9436         # Actualizar label_check_control
9437         label_check_control.config(text="No cumple: Se concluye que el 66% de la
resistencia de la sección de concreto no cumple con el requisito de que la sección
transversal diseñada sea adecuada.")
9438         label_icon_cortante_1.config(image=icon_check_on)
9439
9440 def Revisar_condiciones_diseño():
9441     try:
9442         # Obtener los valores de los Entry, ignorando el texto "kN"
9443         Vu_ultimo_text = entry_Vu_ultimo.get()
9444         φVc_revision_text = entry_φVc_revision.get()
9445         φVcmedios_revision_text = entry_φVcmedios_revision.get()
9446
9447         # Usamos expresiones regulares para extraer solo los valores numéricos
antes de "kN"
9448         Vu_ultimo = float(re.match(r"([0-9.]+)", Vu_ultimo_text).group(1))
9449         φVc_revision = float(re.match(r"([0-9.]+)", φVc_revision_text).group(1))
9450         φVcmedios_revision = float(re.match(r"([0-9.]+)",
φVcmedios_revision_text).group(1))
9451
9452     except ValueError:
9453         print("Error: Asegúrese de que todos los campos tengan valores numéricos
válidos.")
9454         return
9455     except AttributeError:
9456         print("Error: No se pudo extraer el valor numérico de uno de los campos.")
9457         return
9458
9459     # Verificar las condiciones y realizar las comparaciones
9460     if Vu_ultimo > φVc_revision:
9461         # Condición 1: Vu > φVc
9462         check_condición_1.set(1) # Marcar el primer checkbox
9463         check_condición_2.set(0) # Desmarcar el segundo
9464         check_condición_3.set(0) # Desmarcar el tercero

```

```

9465
9466     # Bloquear los demás checkboxes
9467     checkbutton_1_V.config(state=tk.DISABLED)
9468     checkbutton_2_V.config(state=tk.DISABLED)
9469     checkbutton_3_V.config(state=tk.DISABLED)
9470
9471     # Actualizar mensaje
9472     label_mensaje_condicion.config(text="Condición 1:  $V_u > \phi V_c$  [Diseñar
refuerzo transversal]")
9473     label_check_Vs.config(text="Condición de diseño  $V_u > \phi V_c$ : Se debe diseñar
refuerzo transversal")
9474     # Actualizar ícono
9475     label_icon_cortante.config(image=icon_check_on) # Cambiar la imagen del
ícono
9476
9477
9478     elif  $\phi V_{cmedios\_revision} < V_{u\_ultimo} < \phi V_{c\_revision}$ :
9479         # Condición 2:  $\phi V_c/2 < V_u < \phi V_c$ 
9480         check_condición_1.set(0) # Desmarcar el primer checkbox
9481         check_condición_2.set(1) # Marcar el segundo
9482         check_condición_3.set(0) # Desmarcar el tercero
9483
9484         # Bloquear los demás checkboxes
9485         checkbutton_1_V.config(state=tk.DISABLED)
9486         checkbutton_2_V.config(state=tk.DISABLED)
9487         checkbutton_3_V.config(state=tk.DISABLED)
9488
9489         # Actualizar mensaje
9490         label_mensaje_condicion.config(text="Condición 2:  $\phi V_c/2 < V_u < \phi V_c$ 
[Colocar Área mínima]")
9491         label_check_Vs.config(text="Condición de diseño  $\phi V_c/2 < V_u < \phi V_c$ : Colocar
área mínima de refuerzo")
9492         # Actualizar ícono
9493         label_icon_cortante.config(image=icon_check_on) # Cambiar la imagen del
ícono
9494
9495
9496     elif  $V_{u\_ultimo} < \phi V_{cmedios\_revision}$ :
9497         # Condición 3:  $V_u < \phi V_c/2$ 
9498         check_condición_1.set(0) # Desmarcar el primer checkbox
9499         check_condición_2.set(0) # Desmarcar el segundo
9500         check_condición_3.set(1) # Marcar el tercero
9501
9502         # Bloquear los demás checkboxes
9503         checkbutton_1_V.config(state=tk.DISABLED)
9504         checkbutton_2_V.config(state=tk.DISABLED)
9505         checkbutton_3_V.config(state=tk.DISABLED)
9506
9507         # Actualizar mensaje
9508         label_mensaje_condicion.config(text="Condición 3:  $V_u < \phi V_c/2$  [No necesita
refuerzo transversal]")
9509         label_check_Vs.config(text="Condición de diseño  $V_u < \phi V_c/2$ : No necesita
refuerzo transversal")
9510         # Actualizar ícono
9511         label_icon_cortante.config(image=icon_check_on) # Cambiar la imagen del
ícono
9512
9513
9514     # Habilitar el checkbox para revisar cuando se selecciona la opción
9515     check_revisar_widget.config(state=tk.NORMAL)

```

```

9516
9517 def actualizar_label_Smax(*args):
9518     """Actualiza el label Smax_estribos cuando se modifica el valor en
entry_Smax."""
9519     try:
9520         # Obtener el valor del entry_Smax
9521         smax_text = entry_Smax.get().strip()
9522         # Asegurarse de que el valor es un número válido
9523         smax_value = float(re.sub(r'^\d.', '', smax_text)) # Extraer solo el
número
9524         Smax_estribos.config(text=f"{round(smax_value)} cm") # Actualizar el
label con el valor redondeado
9525     except ValueError:
9526         # Si no es un número válido, no actualizar el label
9527         Smax_estribos.config(text="Error: Valor inválido")
9528
9529
9530 def Calcular_Smax():
9531     try:
9532         # Obtener valores de entradas y combobox
9533         fc_revision = combo_fc_revision.get()
9534         texto_resistencia_Vs = entry_resistencia_Vs.get()
9535         resistencia_Vs = float(re.sub(r'^\d.', '', texto_resistencia_Vs)) #
Extraer número
9536
9537         profundidad_prueba = float(label_profundidad_prueba.cget("text"))
9538         b_prueba = float(label_b_prueba.cget("text"))
9539         fc_revision_val = float(fc_revision.split(" ")[0]) # Obtener valor
numérico de fc_revision
9540
9541         # Calcular 33% de la sección
9542         Seccion = 0.33 * math.sqrt(fc_revision_val) * b_prueba *
profundidad_prueba * 1000
9543
9544         # Determinar Smax
9545         Vs_Nsr10 = resistencia_Vs
9546         if Vs_Nsr10 < Seccion:
9547             S_max = profundidad_prueba * 100 / 2
9548         else:
9549             S_max = profundidad_prueba * 100 / 4
9550
9551         # Redondear S_max al número entero más cercano
9552         S_max_redondeado = round(S_max) # Redondear al entero más cercano
9553
9554         # Actualizar campo de entrada y label
9555         entry_Smax.delete(0, tk.END)
9556         entry_Smax.insert(0, f"{S_max_redondeado} cm")
9557
9558         # Actualizar el label Smax_estribos con el valor calculado
9559         Smax_estribos.config(text=f"{S_max_redondeado} cm") # Actualiza el label
9560
9561     except ValueError:
9562         # Manejar errores en entradas
9563         entry_Smax.delete(0, tk.END)
9564         entry_Smax.insert(0, "Error: Entradas inválidas")
9565
9566 def Calcular_Av():
9567     try:
9568         # Obtener los valores de los widgets
9569         texto_Smax = entry_Smax.get()

```

```

9570 Smax = float(re.sub(r'^\d.', '', texto_Smax))
9571 #print(f"Valor de Smax extraído: {Smax}")
9572
9573 b_prueba_texto = label_b_prueba.cget("text")
9574 b_prueba = float(b_prueba_texto)
9575 #print(f"Valor de b_prueba: {b_prueba}")
9576
9577 fc_revision_texto = combo_fc_revision.get()
9578 fc_revision = float(fc_revision_texto.split()[0])
9579 #print(f"Valor de fc_revision: {fc_revision}")
9580
9581 # Definir el valor de fy_Av
9582 fy_Av = 420
9583 #print(f"Valor de fy_Av fijo: {fy_Av}")
9584
9585 # Calcular Sv y b_Av
9586 Sv = Smax * 10
9587 #print(f"Valor de Sv (Smax * 10): {Sv}")
9588
9589 b_Av = b_prueba * 1000
9590 #print(f"Valor de b_Av (b_prueba * 1000): {b_Av}")
9591
9592 # Calcular Av_1 y Av_2
9593 Av_1 = (0.062 * Sv * b_Av * math.sqrt(fc_revision)) / fy_Av
9594 #print(f"Valor de Av_1: {Av_1}")
9595
9596 Av_2 = (0.35 * Sv * b_Av) / fy_Av
9597 #print(f"Valor de Av_2: {Av_2}")
9598
9599 # Seleccionar el valor mayor entre Av_1 y Av_2
9600 resultado = max(Av_1, Av_2)
9601 #print(f"Resultado (máximo entre Av_1 y Av_2): {resultado}")
9602
9603 # Mostrar el resultado en el Entry y en el Label
9604 entry_Av.delete(0, tk.END)
9605 #print("Borrando entry_Av...")
9606
9607 entry_Av.insert(0, f"{resultado:.2f} mm²")
9608 #print("Insertando nuevo valor en entry_Av...")
9609
9610 label_Av_requerida.config(text=f"{resultado:.2f} mm²")
9611 #print("Actualizando label_Av_requerida...")
9612
9613 except ValueError:
9614     print("Error: Los valores deben ser numéricos.")
9615
9616 def Calcular_numero_ramas():
9617     try:
9618         # Obtener los valores de los Entry
9619         valor_Av = entry_Av.get() # Obtener el texto completo
9620         valor_info_area = entry_info_area.get() # Obtener el texto completo
9621
9622         # Usar expresiones regulares para extraer solo los números
9623         Num_Av = float(re.search(r"[-+]?[0-9]*\.?[0-9]+", valor_Av).group()) #
9624         Extraer el primer número
9625         Num_ramas = float(re.search(r"[-+]?[0-9]*\.?[0-9]+",
9626         valor_info_area).group()) # Extraer el primer número
9627
9628         # Realizar el cálculo
9629         if Num_ramas != 0: # Para evitar división por cero

```

```

9628         resultado = Num_Av / Num_ramas
9629     else:
9630         resultado = 0 # Si Num_ramas es 0, el resultado será 0
9631
9632     # Mostrar el resultado en el Entry correspondiente
9633     entry_ramas.delete(0, tk.END) # Borrar cualquier valor previo
9634     entry_ramas.insert(0, str(resultado)) # Insertar el nuevo resultado
9635
9636     # Lógica para aproximar el número de ramas:
9637     # Si el resultado es menor que 2, se asigna 2 ramas.
9638     # Si es mayor o igual a 2, se usa el redondeo hacia arriba como antes.
9639     if resultado < 2:
9640         aproximar_ramas = 2
9641     else:
9642         aproximar_ramas = math.ceil(resultado)
9643
9644     # Actualizar el texto del label_ramas_V con el mensaje de resultados
9645     label_ramas_V.config(text=f"El número de ramas para el estribo calculado
es {resultado:.3f}, "
9646                             f"por lo tanto, se requiere un diseño con
{aproximar_ramas} ramas.")
9647
9648     # Actualizar el texto del label_num_ramas con el valor de aproximar_ramas
9649     label_num_ramas.config(text=f"{aproximar_ramas}")
9650
9651     # Actualizar también el label_check_ramas con el mismo mensaje que
label_ramas_V
9652     label_check_ramas.config(text=f"El número de ramas para el estribo
calculado es {resultado:.3f}, "
9653                             f"por lo tanto, se requiere un diseño con
{aproximar_ramas} ramas.")
9654
9655     except (ValueError, AttributeError):
9656         # Si no se encuentra un número o el formato es incorrecto
9657         entry_ramas.delete(0, tk.END)
9658         entry_ramas.insert(0, "Error: Input inválido")
9659         label_ramas_V.config(text="Error: Input inválido")
9660         label_num_ramas.config(text="Error: Input inválido")
9661
9662
9663 # Función para calcular la resistencia de los estribos
9664 def Calcular_Vs_resistencia_estribos():
9665     try:
9666         # Obtener el valor de entry_info_area (solo números)
9667         valor_info_area = entry_info_area.get() # Obtener el texto completo
9668         Av_r_estribo = float(re.search(r"[-+]?d*\.\d+|\d+",
valor_info_area).group()) # Extraer el primer número
9669
9670         # Obtener el valor de label_profundidad_prueba
9671         d_r_estribo = float(label_profundidad_prueba.cget("text")) # Obtener el
texto y convertirlo a número
9672
9673         # Obtener el valor de entry_Smax (solo números)
9674         valor_Smax = entry_Smax.get() # Obtener el texto completo
9675         S_r_estribo = float(re.search(r"[-+]?d*\.\d+|\d+", valor_Smax).group())
# Extraer el primer número
9676
9677         # Obtener el valor de label_num_ramas (este es num_r_estribo)
9678         # Aquí vamos a asegurarnos de que extraemos solo el número del texto
9679         num_r_estribo_text = label_num_ramas.cget("text")

```

```

9680     num_r_estribo = float(re.search(r"[-+]?[d*\.]\d+|\d+",
num_r_estribo_text).group())) # Extraer el número
9681
9682     # Definir Fy_r_estribo (valor constante)
9683     Fy_r_estribo = 420 # Valor constante dado en el enunciado
9684
9685     # Obtener el valor de entry_info_estribo (solo números)
9686     valor_info_estribo = entry_info_estribo.get() # Obtener el texto completo
9687     Av_info_estribo = float(re.search(r"[-+]?[d*\.]\d+|\d+",
valor_info_estribo).group()))
9688
9689     # Aplicar la fórmula para calcular Vs_resistencia_estribos
9690     Vs_resistencia_estribos = (Fy_r_estribo * num_r_estribo * Av_r_estribo *
d_r_estribo) / S_r_estribo
9691
9692     # Multiplicar el resultado por 0.1
9693     Vs_resistencia_estribos = Vs_resistencia_estribos * 0.1
9694
9695     # Mostrar el resultado en el entry_Vs_estribos
9696     entry_Vs_estribos.delete(0, tk.END) # Borrar cualquier valor previo
9697     entry_Vs_estribos.insert(0, f"{Vs_resistencia_estribos:.3f} kN") #
Insertar el resultado con 2 decimales
9698
9699     # Actualizar el label_Vs_estribos con el mensaje requerido
9700     label_Vs_estribos_1.config(
9701         text=f"El valor calculado de la resistencia de los estribos
{valor_info_estribo} es de {Vs_resistencia_estribos:.3f} kN colocados cada
{S_r_estribo} cm"
9702     )
9703     # Actualizar el label_Vs_resistencia_req con el valor calculado de
Vs_resistencia_estribos
9704     Vs_resistencia_req.config(
9705         text=f"{Vs_resistencia_estribos:.3f} kN"
9706     )
9707
9708
9709     # Actualizar el label_cortante_check_separacion con el mismo mensaje de
label_Vs_estribos_1
9710     Label_check_separacion.config(
9711         text=f"El valor calculado de la resistencia de los estribos
{valor_info_estribo} es de {Vs_resistencia_estribos:.3f} kN colocados cada
{S_r_estribo} cm"
9712     )
9713
9714     except (ValueError, AttributeError):
9715         # Si no se encuentra un número o el formato es incorrecto
9716         entry_Vs_estribos.delete(0, tk.END)
9717         entry_Vs_estribos.insert(0, "Error: Input inválido")
9718         label_Vs_estribos_1.config(text="Error: Input inválido")
9719
9720 # Función para calcular la resistencia requerida Vs
9721 def Calcular_resistencia_Vs_requerida():
9722     try:
9723         # Obtener el valor de entry_Vs_estribos (solo números)
9724         valor_Vs_estribos = entry_Vs_estribos.get() # Obtener el texto completo
9725         Vs_estribos = float(re.search(r"[-+]?[d*\.]\d+|\d+",
valor_Vs_estribos).group())) # Extraer el primer número
9726
9727         # Mostrar el valor de Vs_estribos en entry_resistencia_Vs_requerida

```

```

9728     entry_resistencia_Vs_requerida.delete(0, tk.END) # Borrar cualquier valor
previo
9729     entry_resistencia_Vs_requerida.insert(0, f"{Vs_estribos:.3f} kN") #
Insertar el valor con 3 decimales y unidad
9730
9731     except (ValueError, AttributeError):
9732         # Si no se encuentra un número o el formato es incorrecto
9733         entry_resistencia_Vs_requerida.delete(0, tk.END)
9734         entry_resistencia_Vs_requerida.insert(0, "Error: Input inválido")
9735
9736 def Calcular_resistencia_total_Vn():
9737     try:
9738         # Obtener el valor de entry_resistencia_Vs_requerida
9739         valor_Vs_requerida = entry_resistencia_Vs_requerida.get()
9740         # Usar expresión regular para extraer solo el número
9741         Vs_Vn = float(re.search(r"[-+]?[d*]\.d+|\d+", valor_Vs_requerida).group())
9742
9743         # Obtener el valor de entry_φVc_revision
9744         valor_φVc_revision = entry_φVc_revision.get()
9745         # Extraer solo el número
9746         Vc_Vn = float(re.search(r"[-+]?[d*]\.d+|\d+", valor_φVc_revision).group())
9747
9748         # Obtener directamente el valor de entry_Vu_ultimo y extraer solo el
número
9749         valor_Vu_ultimo = entry_Vu_ultimo.get()
9750         # Usar expresión regular para extraer solo el número
9751         Vu_ultimo = float(re.search(r"[-+]?[d*]\.d+|\d+",
valor_Vu_ultimo).group())
9752
9753         # Obtener el valor de entry_Smax
9754         valor_Smax_req = entry_Smax.get()
9755         Smax_req = float(re.search(r"[-+]?[d*]\.d+|\d+", valor_Smax_req).group())
9756
9757         # Calcular la resistencia total
9758         Resistencia_total_Vn = (0.75 * Vs_Vn) + Vc_Vn
9759
9760         # Comparar Vu_ultimo con Resistencia_total_Vn
9761         if Vu_ultimo <= Resistencia_total_Vn:
9762             cumple_condicion = "Si cumple"
9763         else:
9764             cumple_condicion = "No cumple"
9765
9766         # Buscar si ya existe una fila con el mismo Vu_ultimo
9767         for item in treeview_cortante.get_children():
9768             # Comparar Vu_ultimo con el valor de la segunda columna (que es el
Vu_ultimo)
9769             if float(treeview_cortante.item(item)["values"][1].replace(" kN", ""))
== Vu_ultimo:
9770                 # Si encuentra la fila, actualizar los valores de toda la fila
9771                 treeview_cortante.item(item, values=(
9772                     f"{Vs_Vn:.3f} kN",
9773                     f"{Vu_ultimo:.3f} kN",
9774                     f"{Resistencia_total_Vn:.3f} kN",
9775                     cumple_condicion))
9776                 break # Termina el ciclo una vez que haya actualizado la fila
9777             else:

```

```

9778         # Si no encuentra ninguna fila con el Vu_ultimo, insertar una nueva
fila
9779         treeview_cortante.insert("", "end", values=(
9780             f"{Vs_Vn:.3f} kN",
9781             f"{Vu_ultimo:.3f} kN",
9782             f"{Resistencia_total_Vn:.3f} kN",
9783             cumple_condicion))
9784
9785         # Actualizar el label_resistencia_total
9786         label_resistencia_total.config(
9787             text=f"El valor calculado de  $\phi V_n$  es de {Resistencia_total_Vn:.3f} kN,
lo que representa la capacidad de la sección de concreto cuando se colocan
estribos cada {Smax_req:.2f} cm."
9788         )
9789
9790         # Actualizar el label_φVn con el valor de Resistencia_total_Vn
9791         label_φVn.config(text=f"{Resistencia_total_Vn:.3f} kN")
9792
9793         # Actualizar el label_Vu_φVn con el resultado de cumple_condicion
9794         label_Vu_φVn.config(text=f"{cumple_condicion}")
9795
9796         # **Actualizar el label_check_Vu_φVn con el mensaje completo**
9797         label_check_Vu_φVn.config(
9798             text=f"El valor calculado de  $\phi V_n$  es de {Resistencia_total_Vn:.3f} kN,
lo que representa la capacidad de la sección de concreto cuando se colocan
estribos cada {Smax_req:.2f} cm."
9799         )
9800
9801         except (ValueError, AttributeError) as e:
9802             print(f"Error: {e}") # Solo para depuración si es necesario
9803             print("Error: Input inválido")
9804
9805
9806 def Insertar_datos_cortante_diseño():
9807     # Obtener los valores de los labels
9808     Vmax = label_Vmax_prueba.cget("text").strip() # Vmax de label_Vmax_prueba
9809     Vu = label_Vu.cget("text").strip() # Vu de label_Vu
9810     φVc = label_φVc.cget("text").strip() # φVc de label_φVc
9811     φVc_2 = label_φVc_2.cget("text").strip() # φVc/2 de label_φVc_2
9812     ancho_columna = label_ancho_columna.cget("text").strip() # Ancho de columna
de label_ancho_columna
9813
9814     # Verificar si los valores no están vacíos
9815     if not Vmax or not Vu or not φVc or not φVc_2 or not ancho_columna:
9816         print("Error: Algunos de los valores están vacíos.")
9817         return
9818
9819     # Comprobar si ya hay una fila existente y actualizarla
9820     if treeview_cortante_valores.get_children():
9821         # Tomamos la primera fila (podrías elegir otra fila si es necesario)
9822         item_id = treeview_cortante_valores.get_children()[0] # Primer ítem
(fila)
9823         treeview_cortante_valores.item(item_id, values=(Vmax, Vu, φVc, φVc_2)) #
Actualizamos la fila existente
9824     else:
9825         # Si no hay filas, insertamos la primera fila
9826         treeview_cortante_valores.insert("", tk.END, values=(Vmax, Vu, φVc,
φVc_2))
9827
9828     # Convertir el valor de ancho de columna de cm a metros

```

```
9829     try:
9830         ancho_columna_float = float(ancho_columna.replace("cm", "").strip()) #
Eliminar "cm" y convertir a float
9831         ancho_columna_metros = ancho_columna_float / 100 # Convertir de cm a
metros
9832     except ValueError:
9833         print("Error: El valor de ancho de columna no es numérico.")
9834         return
9835
9836     # Mostrar el valor convertido en el Entry de ancho de columna en metros
9837     entry_ancho_C.delete(0, tk.END) # Limpiar el Entry antes de insertar el nuevo
valor
9838     entry_ancho_C.insert(0, f"{ancho_columna_metros:.2f} m") # Insertar el valor
en metros
9839
9840     # Calcular la mitad del ancho de la columna en metros
9841     mitad_ancho = ancho_columna_metros / 2 # Calcular la mitad
9842
9843     # Actualizar el Entry con la mitad del ancho de la columna
9844     entry_borde_C.delete(0, tk.END) # Limpiar el Entry antes de insertar el nuevo
valor
9845     entry_borde_C.insert(0, f"{mitad_ancho:.2f} m") # Insertar el valor de la
mitad en metros
9846
9847
9848 def dibujar_longitud_triangulo_cortante():
9849     # Obtener los valores de las variables
9850
9851     try:
9852         Wu_cortante_C = float(label_seccion_Wu.cget("text"))
9853         L_cortante_C = float(label_seccion_luz.cget("text"))
9854         label_profundidad_prueba_valor =
float(label_profundidad_prueba.cget("text"))
9855     except ValueError:
9856         print("Error: Los valores de Wu_cortante_C, L_cortante_C y
label_profundidad_prueba deben ser numéricos.")
9857         return
9858
9859     # Calcular las reacciones en los apoyos
9860     Reaccion_cortante_apoyos_C = (Wu_cortante_C * L_cortante_C) / 2
9861     Ray_cortante_C = Reaccion_cortante_apoyos_C
9862     Rby_cortante_C = Ray_cortante_C
9863
9864     # Asignar Vmax y base_l de acuerdo con la nueva lógica
9865     Vmax = Ray_cortante_C
9866     base_l = L_cortante_C / 2
9867
9868
9869     # Actualizar el valor en el Entry
9870     entry_base_triangulo.delete(0, tk.END) # Borrar el valor anterior
9871     entry_base_triangulo.insert(0, str(base_l)) # Insertar el nuevo valor
9872
9873     # Crear una figura y un eje para graficar el triángulo
9874     fig, ax = plt.subplots()
9875
9876     # Desplazar el triángulo más a la derecha ajustando las coordenadas
9877     desplazamiento = 2
9878
9879     # Factor de escala para hacer el triángulo más grande
9880     factor_escal = 1.8
```

```
9881
9882     # Definir las coordenadas del triángulo (x, y)
9883     x = [base_l * factor_escal + desplazamiento, 0 + desplazamiento, 0 +
desplazamiento]
9884     y = [0, 0, Vmax * factor_escal]
9885
9886     # Dibujar el triángulo conectando los puntos
9887     ax.fill(x, y, color=(0.5, 0.5, 0.5, 0.3))
9888     ax.plot(x + [x[0]], y + [y[0]], color='black')
9889
9890     # Configurar los límites del gráfico para mostrar solo el triángulo positivo
9891     ax.set_xlim(-1, base_l * factor_escal + desplazamiento + 1)
9892     ax.set_ylim(-1, Vmax * factor_escal * 1.05 + 1)
9893
9894     # Habilitar los grids
9895     ax.grid(True)
9896
9897     # Remover los títulos de los ejes y las marcas (ticks)
9898     ax.set_xticks([])
9899     ax.set_yticks([])
9900     ax.set_xlabel('')
9901     ax.set_ylabel('')
9902
9903     # Remover el borde de los ejes (esto elimina el borde negro alrededor de la
gráfica)
9904     ax.spines['top'].set_visible(False)
9905     ax.spines['right'].set_visible(False)
9906     ax.spines['bottom'].set_visible(False)
9907     ax.spines['left'].set_visible(False)
9908
9909     # Agregar marcador rojo en la altura (Vmax) del triángulo
9910     marcador_rojo=ax.plot(0 + desplazamiento, Vmax * factor_escal,
'ro',markersize=4)
9911
9912     # Añadir la etiqueta con el valor de Vmax al lado izquierdo del marcador
9913     ax.text(0 + desplazamiento + 0.2, Vmax * factor_escal, f"Vmax", color='r',
fontsize=8, verticalalignment='center')
9914
9915     # --- NUEVO CÁLCULO PARA LA CORTANTE EN x ---
9916     try:
9917         # Obtener el valor de x desde el Entry (ignorar texto y tomar el valor
numérico)
9918         valor_entry = entry_borde_C.get()
9919
9920         # Extraer solo el valor numérico usando expresiones regulares
9921         x_cortante_match = re.findall(r"[-+]?d*\.\d+|\d+", valor_entry)
9922
9923         if x_cortante_match:
9924             x_cortante = float(x_cortante_match[0])
9925         else:
9926             print("Error: No se encontró un valor numérico válido en el Entry.")
9927             return
9928
9929         # Validar que x_cortante esté dentro del rango válido
9930         if x_cortante < 0 or x_cortante > base_l:
9931             print("Error: El valor de x debe estar entre 0 y la base de la viga.")
9932             return
9933
9934         # Calcular el valor de la cortante en x usando la fórmula del triángulo
9935         cortante_en_x = Vmax * (1 - (x_cortante / base_l))
```

```

9936
9937     # Añadir marcador en el gráfico para el valor de cortante en x
9938     marcador_rojo_cortante= ax.plot(x_cortante * factor_escala +
desplazamiento, cortante_en_x * factor_escala, 'ro',markersize=4)
9939     ax.text(x_cortante * factor_escala + desplazamiento +0.2, cortante_en_x *
factor_escala,
9940             f"V(borde)", color='r', fontsize=8)
9941
9942     # --- Agregar la línea discontinua roja en Vmax (marcador) ---
9943     ax.plot([0 + desplazamiento, 0 + desplazamiento], [0, Vmax *
factor_escala], 'r--')
9944
9945     # --- Agregar la línea discontinua roja en cortante en x (marcador) ---
9946     ax.plot([x_cortante * factor_escala + desplazamiento, x_cortante *
factor_escala + desplazamiento],
9947             [0, cortante_en_x * factor_escala], 'r--')
9948
9949     # --- Agregar la línea discontinua horizontal roja en cortante en x ---
9950     ax.plot([0 + desplazamiento, x_cortante * factor_escala + desplazamiento],
9951             [cortante_en_x * factor_escala, cortante_en_x * factor_escala],
'r--')
9952
9953     # --- Cálculo para Vu_base y posición de Vu ---
9954     # Obtener el valor de entry_Vu_ultimo desde el Entry
9955     valor_entry_Vu_ultimo = entry_Vu_ultimo.get()
9956
9957     # Extraer solo el valor numérico usando expresiones regulares
9958     x_Vu_match = re.findall(r"[-+]?[d*]\.d+|\d+", valor_entry_Vu_ultimo)
9959
9960     if x_Vu_match:
9961         Vu_ultimo = float(x_Vu_match[0])
9962     else:
9963         print("Error: No se encontró un valor numérico válido en el Entry
Vu_ultimo.")
9964         return
9965
9966     # Paso 1: Calcular Vu_base
9967     Vu_base = Vu_ultimo * base_l
9968
9969
9970     # Paso 2: Despejar x_Vu
9971     x_Vu = Vu_base / Vmax
9972
9973
9974     # Paso 3: Calcular (x)_b_Vu
9975     x_b_Vu = base_l - x_Vu
9976
9977
9978     # Paso 4: Calcular (x)_c_Vu
9979     x_c_Vu = x_b_Vu
9980
9981
9982     # Ubicar el marcador morado en la posición correcta de Vu
9983     marcador_Vu_morado = ax.plot(x_c_Vu * factor_escala + desplazamiento,
Vu_ultimo * factor_escala, 'mo', markersize=4)
9984
9985     # Añadir la etiqueta con el valor de Vu al lado del marcador
9986     ax.text(x_c_Vu * factor_escala + desplazamiento + 0.2, Vu_ultimo *
factor_escala +2 ,
9987             f"Vu", color='m', fontsize=8)

```

```

9988
9989     # --- Agregar la línea discontinua morada en Vu (vertical) ---
9990     ax.plot([x_c_Vu * factor_escala + desplazamiento, x_c_Vu * factor_escala +
desplazamiento],
9991             [0, Vu_ultimo * factor_escala], 'm--')
9992
9993     # --- Agregar la línea discontinua morada horizontal ---
9994     ax.plot([0 + desplazamiento, x_c_Vu * factor_escala + desplazamiento],
9995             [Vu_ultimo * factor_escala, Vu_ultimo * factor_escala], 'm--')
9996
9997     # --- NUEVO CÁLCULO PARA  $\phi Vc_{revision}$  ---
9998
9999     # Obtener el valor de  $\phi Vc_{revision}$  desde el Entry
10000     valor_entry_φVc_revision = entry_φVc_revision.get()
10001
10002     # Extraer solo el valor numérico usando expresiones regulares
10003     x_φVc_revision_match = re.findall(r"[-+]?[0-9]*\.?[0-9]+",
valor_entry_φVc_revision)
10004
10005     if x_φVc_revision_match:
10006         cortante_φVc_revision = float(x_φVc_revision_match[0])
10007     else:
10008         print("Error: No se encontró un valor numérico válido en el Entry
φVc_revision.")
10009         return
10010
10011     # Paso 1: Calcular  $\phi Vc_{base}$ 
10012     φVc_base = cortante_φVc_revision * base_1
10013
10014
10015     # Paso 2: Despejar x
10016     x = φVc_base / Vmax
10017
10018
10019     # Paso 3: Calcular (x)_1
10020     x_1 = base_1 - x
10021
10022
10023     # Paso 4: Calcular (x)_2
10024     x_2 = x_1
10025
10026
10027     # Ubicar el marcador verde en la posición correcta de  $\phi Vc_{revision}$ 
10028     marcador_verde = ax.plot(x_2 * factor_escala + desplazamiento,
cortante_φVc_revision * factor_escala, 'go', markersize=4)
10029
10030     # Añadir la etiqueta con el valor de la cortante  $\phi Vc_{revision}$  al lado del
marcador verde
10031     ax.text(x_2 * factor_escala + desplazamiento + 0.2, cortante_φVc_revision *
factor_escala - 6,
10032            f"φVc", color='g', fontsize=8)
10033
10034     # Agregar una línea discontinua verde en el marcador de cortante
φVc_revision (vertical)
10035     ax.plot([x_2 * factor_escala + desplazamiento, x_2 * factor_escala +
desplazamiento],
10036            [0, cortante_φVc_revision * factor_escala], 'g--')
10037
10038     # Agregar la línea discontinua verde horizontal en cortante  $\phi Vc_{revision}$ 
10039     ax.plot([0 + desplazamiento, x_2 * factor_escala + desplazamiento],

```

```

10040         [cortante_φVc_revision * factor_escala, cortante_φVc_revision *
factor_escala], 'g--')
10041
10042
10043         # --- NUEVO CÁLCULO PARA φVcmedios_revision ---
10044
10045         # Obtener el valor de φVcmedios_revision desde el Entry
10046         valor_entry_φVcmedios_revision = entry_φVcmedios_revision.get()
10047
10048         # Extraer solo el valor numérico usando expresiones regulares
10049         x_φVcmedios_revision_match = re.findall(r"[-+]?[0-9]*\.?[0-9]+",
valor_entry_φVcmedios_revision)
10050
10051         if x_φVcmedios_revision_match:
10052             cortante_φVcmedios_revision = float(x_φVcmedios_revision_match[0])
10053         else:
10054             print("Error: No se encontró un valor numérico válido en el Entry
φVcmedios_revision.")
10055             return
10056
10057         # Paso 1: Calcular φVc_medios_base
10058         φVc_base_medios = cortante_φVcmedios_revision * base_l
10059
10060
10061         # Paso 2: Despejar x_a
10062         x_a = φVc_base_medios / Vmax
10063
10064
10065         # Paso 3: Calcular (x)_b
10066         x_b = base_l - x_a
10067
10068
10069         # Paso 4: Calcular (x)_c
10070         x_c = x_b
10071
10072
10073         # Ubicar el marcador verde en la posición correcta de φVcmedios_revision
10074         marcador_azul = ax.plot(x_c * factor_escala + desplazamiento,
cortante_φVcmedios_revision * factor_escala, 'bo', markersize=4)
10075
10076         # Añadir la etiqueta con el valor de la cortante φVcmedios_revision al
lado del marcador verde
10077         ax.text(x_c * factor_escala + desplazamiento + 0.2,
cortante_φVcmedios_revision * factor_escala + 2,
10078             f"φVc/2", color='b', fontsize=8)
10079
10080         # Agregar una línea discontinua verde en el marcador de cortante
φVcmedios_revision (vertical)
10081         ax.plot([x_c * factor_escala + desplazamiento, x_c * factor_escala +
desplazamiento],
10082             [0, cortante_φVcmedios_revision * factor_escala], 'b--')
10083
10084         # Agregar la línea discontinua verde horizontal en cortante
φVcmedios_revision
10085         ax.plot([0 + desplazamiento, x_c * factor_escala + desplazamiento],
10086             [cortante_φVcmedios_revision * factor_escala, cortante_φVcmedios_revision
* factor_escala], 'b--')
10087
10088         # --- NUEVO CÁLCULO PARA Vs_estribos ---
10089

```

```

10090     # Obtener el valor de Vs_estribos desde el Entry
10091     valor_entry_Vs_estribos = entry_Vs_estribos.get()
10092
10093     # Extraer solo el valor numérico usando expresiones regulares
10094     x_Vs_estribos_match = re.findall(r"[+]?\d*\.\d+|\d+",
valor_entry_Vs_estribos)
10095
10096     if x_Vs_estribos_match:
10097         cortante_Vs_estribos = float(x_Vs_estribos_match[0])
10098     else:
10099         print("Error: No se encontró un valor numérico válido en el Entry
Vs_estribos.")
10100         return
10101
10102     # Paso 1: Calcular Vs_estribos_base
10103     Vs_estribos_base = cortante_Vs_estribos * base_l
10104
10105
10106     # Paso 2: Despejar x_estribo
10107     x_estribo = Vs_estribos_base / Vmax
10108
10109
10110     # Paso 3: Calcular (x)_estribo_b
10111     x_estribo_b = base_l - x_estribo
10112
10113
10114     # Paso 4: Calcular (x)_estribo_c
10115     x_estribo_c = x_estribo_b
10116
10117
10118     # Ubicar el marcador amarillo en la posición correcta de Vs_estribos
10119     marcador_naranja = ax.plot(x_estribo_c * factor_escal + desplazamiento,
cortante_Vs_estribos * factor_escal, 'o', color=(1.0, 0.647, 0.0), markersize=4)
10120
10121     # Añadir la etiqueta con el valor de la cortante Vs_estribos al lado del
marcador amarillo
10122     ax.text(x_estribo_c * factor_escal + desplazamiento + 0.2,
cortante_Vs_estribos * factor_escal + 8,
10123           f"Vs", color=(1.0, 0.647, 0.0), fontsize=8)
10124
10125     # Agregar una línea discontinua amarilla en el marcador de cortante
Vs_estribos (vertical)
10126     ax.plot([x_estribo_c * factor_escal + desplazamiento, x_estribo_c *
factor_escal + desplazamiento],
10127           [0, cortante_Vs_estribos * factor_escal], '--',color=(1.0, 0.647, 0.0))
10128
10129     # Agregar la línea discontinua amarilla horizontal en cortante Vs_estribos
10130     ax.plot([0 + desplazamiento, x_estribo_c * factor_escal +
desplazamiento],
10131           [cortante_Vs_estribos * factor_escal, cortante_Vs_estribos *
factor_escal], '--',color=(1.0, 0.647, 0.0))
10132
10133
10134     # Agregar leyenda para el marcador rojo con el valor de Vmax
10135     ax.legend(
10136         [marcador_rojo[0], marcador_rojo_cortante[0], marcador_Vu_morado[0],
marcador_verde[0], marcador_azul[0], marcador_naranja[0]],
10137         [f"Vmax: {Vmax} kN", f"V(borde): {cortante_en_x:.2f} kN", f"Vu:
{Vu_ultimo:.2f} kN",

```

```

10138         f"φVc: {cortante_φVc_revision:.2f} kN", f"φVc/2:
{cortante_φVcmedios_revision:.2f} kN", f"Vs: {cortante_Vs_estribos:.2f} kN"],
10139         loc='upper left', fontsize=7, bbox_to_anchor=(-0.17, 1),
10140         frameon=False,
10141     )
10142
10143     # Crear la anotación FIJA a la derecha del gráfico sin borde
10144     anotacion = ax.text(
10145         L_cortante_C * 1.1, Vmax * 1.5,
10146         f"Valores\nV: {Vmax:.2f} kN\nL: {0:.2f} m",
10147         color="black", fontsize=7, ha="left",
10148         bbox=dict(facecolor='white', boxstyle='round,pad=0.5', edgecolor='none')
10149     )
10150
10151     except ValueError:
10152         print("Error: Ingrese un valor numérico válido para x.")
10153         return
10154
10155     # Comprobar si el Checkbutton está marcado (comprobamos la variable
BooleanVar)
10156     if check_longitud_refuerzo_var.get():
10157         # Colocar la figura en el canvas de tkinter
10158         canvas_1 = FigureCanvasTkAgg(fig, master=canvas_longitud)
10159         canvas_1.draw()
10160         canvas_1.get_tk_widget().pack(fill="both", expand=True)
10161     else:
10162         # Si no está marcado, borrar el canvas
10163         for widget in canvas_longitud.winfo_children():
10164             widget.destroy()
10165
10166     # Guardar la imagen en alta calidad (DPI 300)
10167     ruta_guardado_2 = os.path.join("Reportes", "imagenes_pilot",
"longitud_refuerzo.png")
10168     os.makedirs(os.path.dirname(ruta_guardado_2), exist_ok=True) # Crear
directorio si no existe
10169     fig.savefig(ruta_guardado_2, dpi=300, bbox_inches='tight')
10170
10171
10172     # Crear el Slider
10173     slider_cortante = tk.Scale(labelframe_estribos, from_=0, to=base_l,
orient="horizontal", resolution=0.00)
10174     slider_cortante.set(base_l) # Establecer el valor máximo
10175     slider_cortante.place(x=250, y=1)
10176
10177     # Crear entry_posicion
10178     entry_posicion = tk.Entry(labelframe_estribos)
10179     entry_posicion.place(x=90,y=10,width=80)
10180
10181     # Crear el label para mostrar el valor del slider
10182     label_valor = tk.Label(labelframe_estribos, text=f"x={base_l:.2f}
m",bd=2,relief="ridge")
10183     label_valor.place(x=253,y=1,width=100)
10184
10185     # Crear el marcador negro, iniciando en la posición del vértice superior
(Vmax)
10186     marcador_negro, = ax.plot([desplazamiento], [Vmax * factor_escala], 'ko',
markersize=4)
10187
10188     linea_discontinua, = ax.plot([desplazamiento, desplazamiento], [Vmax *
factor_escala, 0], 'k--')

```

```

10189
10190
10191     # Función para redimensionar la imagen
10192     def redimensionar_imagen(ruta_imagen, tamano):
10193         imagen_agregar_eliminar = Image.open(ruta_imagen) # Abrir la imagen
10194         imagen_agregar_eliminar = imagen_agregar_eliminar.resize(tamano) #
Redimensionarla
10195         return ImageTk.PhotoImage(imagen_agregar_eliminar) # Convertirla en un
formato compatible con Tkinter
10196
10197     # Cargar las imágenes de los iconos redimensionadas
10198     imagen_agregar =
redimensionar_imagen("Iconos_proyecto/agregar_marcador.png", (20, 20))
10199     imagen_eliminar =
redimensionar_imagen("Iconos_proyecto/eliminar_marcador.png", (20, 20))
10200
10201     # Mantener una referencia de las imágenes
10202     boton_agregar_marcador.icono = imagen_agregar
10203     boton_eliminar_marcador.icono = imagen_eliminar
10204
10205
10206     def actualizar_marcador(value):
10207
10208         # Obtener el valor ingresado en el Entry
10209         try:
10210             valor_entry = float(entry_posicion.get()) # Capturamos el dato del
Entry
10211         except ValueError:
10212             print("Error: El valor ingresado en entry_posicion no es válido.")
10213             return
10214
10215         nuevo_x = float(value) * factor_escala + desplazamiento
10216         nuevo_y = Vmax * (1 - (float(value) / base_l))
10217
10218         marcador_negro.set_data([nuevo_x], [nuevo_y * factor_escala])
10219
10220         # Actualizar la línea discontinua para que llegue hasta la base
10221         linea_discontinua.set_data([nuevo_x, nuevo_x], [nuevo_y * factor_escala,
0])
10222
10223         anotacion.set_text(f"Valores\nV: {nuevo_y:.2f} kN\nL: {value:.2f} m")
10224
10225         # Actualizar el texto del label con el formato "x={valor:.2f}"
10226         label_valor.config(text=f"x={float(value):.2f} m")
10227
10228         canvas_1.draw()
10229
10230
10231     # Asociar el evento del slider con la función de actualización
10232     slider_cortante.configure(command=lambda val: actualizar_marcador(float(val)))
10233
10234     # Función para que el Slider se actualice con el valor del Entry
10235     def actualizar_slider_desde_entry():
10236         try:
10237             # Obtener el valor del Entry
10238             valor_entry = float(entry_posicion.get())
10239
10240             # Asegurarse de que esté dentro del rango
10241             if 0 <= valor_entry <= base_l:
10242                 # Actualizar el Slider y el marcador

```

```

10243         slider_cortante.set(valor_entry)
10244         actualizar_marcador(valor_entry) # Actualizar el gráfico
10245     else:
10246         print("Valor fuera de rango.")
10247 except ValueError:
10248     print("Por favor ingrese un valor numérico válido en el Entry.")
10249
10250 # Función que permite cambiar el valor del Entry al mover el Slider
10251 def actualizar_entry_desde_slider(value):
10252     entry_posicion.delete(0, tk.END)
10253     entry_posicion.insert(0, str(value))
10254
10255 # Asignar la función para actualizar el marcador cuando el usuario presiona
Enter
10256     entry_posicion.bind("<Return>", lambda event: actualizar_slider_desde_entry())
10257
10258 # Función que permite cambiar el valor del Entry al mover el Slider
10259 def actualizar_entry_desde_slider(value):
10260     entry_posicion.delete(0, tk.END)
10261     entry_posicion.insert(0, str(value)) # Actualizar el valor del Entry
cuando el Slider cambia
10262
10263
10264 def agregar_marcador():
10265     """Función para agregar una nueva línea discontinua al hacer clic en el
botón"""
10266     global ultimo_valor_borde_C
10267     try:
10268         # Capturamos el valor de entry_posicion
10269         valor_entry = float(entry_posicion.get()) # Capturar el dato del
Entry
10270         # Capturamos el valor de entry_borde_C y limpiamos cualquier texto
adicional
10271         valor_borde_C_str = entry_borde_C.get() # Capturamos el valor de
entry_borde_C como string
10272         valor_borde_C = float(re.sub(r'^0-9.', '', valor_borde_C_str)) #
Elimina todo lo que no sea número o punto decimal
10273         valor_borde_C = valor_borde_C + 0.05 # Sumamos 0.05 al valor obtenido
de entry_borde_C
10274     except ValueError:
10275         print("Error: El valor ingresado en entry_posicion o entry_borde_C no
es válido.")
10276     return
10277
10278     nuevo_x = valor_entry * factor_escala + desplazamiento
10279     nuevo_y = Vmax * (1 - (valor_entry / base_l))
10280
10281 # Crear una nueva línea discontinua
10282     nueva_linea, = ax.plot([nuevo_x, nuevo_x], [nuevo_y * factor_escala, 0],
'k-', linewidth=1.5)
10283
10284 # Guardar la línea en la lista
10285     lineas_discontinuas.append(nueva_linea)
10286     posiciones_x.append(valor_entry)
10287
10288 # Calcular la distancia entre las últimas dos líneas si hay más de una
10289     if len(posiciones_x) > 1:
10290         distancia = posiciones_x[-1] - posiciones_x[-2]
10291
10292

```

```

10293     # Posiciones en x para la cota
10294         x1 = posiciones_x[-2] * factor_escala + desplazamiento
10295         x2 = posiciones_x[-1] * factor_escala + desplazamiento
10296         y_cota = -17
10297
10298     # Línea de la cota
10299         linea_cota, = ax.plot([x1, x2], [y_cota, y_cota], 'r-', linewidth=1.5)
10300
10301     # Marcas de los extremos de la cota
10302         marca_izq, = ax.plot([x1, x1], [y_cota - 2, y_cota + 2], 'r-',
10303 linewidth=1.5)
10304         marca_der, = ax.plot([x2, x2], [y_cota - 2, y_cota + 2], 'r-',
10305 linewidth=1.5)
10306
10307     # Texto con el valor de la distancia
10308         texto_cota = ax.text((x1 + x2) / 2, y_cota - 3, f"{distancia:.2f}",
10309 color='black', fontsize=7, ha='center')
10310         textos_cotas.append(texto_cota) # Guardamos el texto en la lista
10311
10312     # Comprobamos si el valor de entry_borde_C ha cambiado
10313     if valor_borde_C != ultimo_valor_borde_C:
10314         ultimo_valor_borde_C = valor_borde_C
10315
10316     # Calculamos la posición en X de la cota adicional
10317         nuevo_x_borde = valor_borde_C * factor_escala + desplazamiento
10318         y_cota_borde = y_cota # Usamos la misma posición Y para la cota
10319 adicional
10320
10321     # Línea de la cota adicional (verde y discontinua)
10322         linea_cota_borde, = ax.plot([nuevo_x_borde, nuevo_x_borde],
10323 [y_cota_borde, y_cota_borde + 5], 'g--', linewidth=1.5)
10324
10325     # Marcas de los extremos de la cota adicional
10326         marca_izq_borde, = ax.plot([nuevo_x_borde, nuevo_x_borde],
10327 [y_cota_borde - 2, y_cota_borde + 2], 'g-', linewidth=1.5)
10328         marca_der_borde, = ax.plot([nuevo_x_borde, nuevo_x_borde],
10329 [y_cota_borde - 2, y_cota_borde + 2], 'g-', linewidth=1.5)
10330
10331     # Texto con el valor de la cota adicional
10332         texto_cota_borde = ax.text(nuevo_x_borde, y_cota_borde - 3, f"
10333 {valor_borde_C:.2f}", color='black', fontsize=7, ha='center')
10334         textos_cotas.append(texto_cota_borde) # Guardamos el texto en la
10335 lista
10336
10337     # Guardar ambas cotas (principal y adicional) en la lista de cotas
10338         cotas.append((linea_cota, marca_izq, marca_der, texto_cota,
10339 linea_cota_borde, marca_izq_borde, marca_der_borde, texto_cota_borde))
10340
10341     # Redibujar el canvas
10342         canvas_1.draw()
10343
10344 def borrar_lineas_discontinuas():
10345     """Función para borrar todas las líneas discontinuas del gráfico."""
10346     global lineas_discontinuas, posiciones_x, cotas, textos_cotas
10347
10348     # Eliminar todas las líneas discontinuas del gráfico
10349     for linea in lineas_discontinuas:
10350         linea.remove() # Removemos cada línea del gráfico

```

```
10343
10344     # Limpiar las listas
10345     lineas_discontinuas.clear()
10346     posiciones_x.clear()
10347
10348     # Eliminar todas las cotas del gráfico
10349     for cota in cotas:
10350         for elemento in cota: # Cada cota contiene la línea, las marcas y el
10351             texto
10352                 if elemento in ax.get_children(): # Verificamos si el elemento
10353                     aún está en el gráfico
10354                         elemento.remove()
10355
10356     # Limpiar la lista de cotas
10357     cotas.clear()
10358
10359     # Eliminar los textos de las cotas
10360     for texto in textos_cotas:
10361         if texto in ax.get_children(): # Verificamos si el texto aún está en
10362             el gráfico
10363                 texto.remove() # Removemos cada objeto de texto
10364
10365     # Limpiar la lista de textos
10366     textos_cotas.clear()
10367
10368     # Redibujar el canvas sin las líneas y textos
10369     canvas_1.draw()
10370
10371     # Crear los botones con la nueva lógica
10372     boton_agregar_marcador = tk.Button(labelframe_estribos, image=imagen_agregar,
10373         bd=2, relief="flat",command=agregar_marcador)
10374     boton_agregar_marcador.place(x=180, y=10)
10375
10376     boton_eliminar_marcador = tk.Button(labelframe_estribos,
10377         image=imagen_eliminar, bd=2, relief="flat",command=borrar_lineas_discontinuas)
10378     boton_eliminar_marcador.place(x=215, y=10)
10379
10380 def agregar_datos_treeview_cortante():
10381     """Obtiene los datos de los labels y los inserta en el treeview sin duplicar
10382     filas."""
10383     # Obtener los valores de los labels
10384     wu = label_seccion_wu.cget("text") + "kN/m"
10385     vmax = label_Vmax_prueba.cget("text")
10386     vu = label_Vu.cget("text")
10387     vc = label_φVc.cget("text")
10388     vc_2 = label_φVc_2.cget("text")
10389     vs = Vs_resistencia.cget("text")
10390
10391     # Obtener el valor del label_check_Vs
10392     check_vs = label_check_Vs.cget("text")
10393     check_control = label_check_control.cget("text")
10394
10395     # Verificar si ya hay una fila en el primer treeview
10396     items_1 = treeview_cortante_1.get_children()
10397     if items_1:
10398         # Actualizar la primera fila existente
```

```

10397     treeview_cortante_1.item(items_1[0], values=(wu, vmax, vu, vc, vc_2, vs))
10398 else:
10399     # Insertar una nueva fila si está vacío
10400     treeview_cortante_1.insert("", "end", values=(wu, vmax, vu, vc, vc_2, vs))
10401
10402     # Actualizar información
10403     label_cortante_check_Vs.config(text=check_vs)
10404     label_control_check.config(text=check_control)
10405
10406     # Obtener los valores de los otros labels para el segundo Treeview
10407     Smax_estribos_text = Smax_estribos.cget("text")
10408     Av_requerida_text = label_Av_requerida.cget("text")
10409     num_ramas_text = label_num_ramas.cget("text")
10410     Vs_resistencia_req_text = Vs_resistencia_req.cget("text")
10411
10412     # Verificar si ya hay una fila en el segundo treeview
10413     items_2 = treeview_cortante_2.get_children()
10414     if items_2:
10415         # Actualizar la primera fila existente en treeview_cortante_2
10416         treeview_cortante_2.item(items_2[0], values=(Smax_estribos_text,
10417 Av_requerida_text, num_ramas_text, Vs_resistencia_req_text))
10418     else:
10419         # Insertar una nueva fila si está vacío
10420         treeview_cortante_2.insert("", "end", values=(Smax_estribos_text,
10421 Av_requerida_text, num_ramas_text, Vs_resistencia_req_text))
10422
10423     # Obtener el valor de Label_check_ramas y actualizar en
10424     label_cortante_check_ramas
10425     check_ramas = Label_check_ramas.cget("text")
10426     label_cortante_check_ramas.config(text=check_ramas)
10427
10428     # Obtener el valor de Label_check_separacion y actualizar en
10429     label_cortante_check_separacion
10430     check_separacion = Label_check_separacion.cget("text")
10431     label_cortante_check_separacion.config(text=check_separacion)
10432
10433     label_icon_cortante_2.config(image=icon_check_on)
10434
10435     # Obtener los valores de los labels necesarios para treeview_cortante_3
10436     Vs_resistencia_req_text = Vs_resistencia_req.cget("text")
10437     vu_text = label_Vu.cget("text")
10438     label_φVn_text = label_φVn.cget("text") # Valor de label_φVn
10439     label_Vu_φVn_text = label_Vu_φVn.cget("text") # Valor de label_Vu_φVn
10440     label_check_capacidad = label_check_Vu_φVn.cget("text")
10441
10442     # Verificar si ya hay una fila en el tercer treeview (treeview_cortante_3)
10443     items_3 = treeview_cortante_3.get_children()
10444     if items_3:
10445         # Actualizar la primera fila existente en treeview_cortante_3
10446         treeview_cortante_3.item(items_3[0], values=(Vs_resistencia_req_text,
10447 vu_text, label_φVn_text, label_Vu_φVn_text))
10448     else:
10449         # Insertar una nueva fila si está vacío
10450         treeview_cortante_3.insert("", "end", values=(Vs_resistencia_req_text,
10451 vu_text, label_φVn_text, label_Vu_φVn_text))
10452
10453     # Actualizar el label_check_Vu_φVn
10454     label_cortante_capacidad_check.config(text=label_check_capacidad)
10455     label_icon_cortante_3.config(image=icon_check_on)
10456

```

```
10451
10452 def Generar_reportes():
10453     # Obtener el directorio de trabajo actual (directorio donde está el script)
10454     current_dir = os.path.dirname(os.path.realpath(__file__)) # Directorio donde
se ejecuta el script
10455
10456     # Definir rutas relativas basadas en el directorio actual
10457     doc_template_path = os.path.join(current_dir, 'Reportes',
'Formato_reporte.docx')
10458
10459     # Obtener los datos de la interfaz de usuario
10460     nombre_estudiante = label_estudiante.cget("text")
10461     nombre_modelo = label_modelo.cget("text")
10462     parametro_fc = label_fc.cget("text")
10463     parametro_Ec = label_elasticidad.cget("text")
10464     parametro_viga = label_seccion_cargas.cget("text")
10465     parametro_altura = label_h.cget("text")
10466     parametro_concreto = label_resistencia_psi.cget("text")
10467     parametro_Wpp = label_cargas_Wpp.cget("text")
10468     parametro_WD = label_cargas_WD.cget("text")
10469     parametro_WL = label_cargas_WL.cget("text")
10470     parametro_Wtotal = label_cargas_Wtotal.cget("text")
10471     parametro_Wu = label_seccion_Wu.cget("text")
10472     parametro_luz = label_seccion_luz.cget("text")
10473     parametro_tipo_viga = label_tipo_viga_cargas.cget("text")
10474     parametro_apoyos = label_reaccion.cget("text")
10475     parametro_Vmax = label_reaccion.cget("text")
10476     parametro_Mmax = label_Mmax_1.cget("text")
10477     parametro_Mmax_2 = label_Mmax_2.cget("text")
10478     parametro_cuantia_min = label_cuantia_minima.cget("text")
10479     parametro_cuantia_max = label_cuantia_maxima.cget("text")
10480     parametro_beta1 = label_beta1.cget("text")
10481     parametro_Rn = label_Resistencia_Rn.cget("text")
10482     parametro_cuantia_req = label_cuantia_requerida.cget("text")
10483     parametro_acero_req = label_acero_requerido.cget("text")
10484     parametro_Rn2 = label_Rn2.cget("text")
10485     parametro_C2 = label_req_C2.cget("text")
10486     parametro_As2 = label_As2.cget("text")
10487     parametro_estribo = label_estribo.cget("text")
10488     parametro_area_estribo = label_area_estribo.cget("text")
10489     parametro_diametro_estribo = label_prueba_estribo.cget("text")
10490     parametro_φMn1 = label_φMn1.cget("text")
10491     parametro_φMn2 = label_φMn2.cget("text")
10492     parametro_conclusion1 = label_conclusion_φMn.cget("text")
10493     parametro_conclusion2 = label_conclusion_φMn2.cget("text")
10494     parametro_check = label_check.cget("text")
10495     parametro_No_barra= label_barra.cget("text")
10496     parametro_area_barra = label_area_barra.cget("text")
10497     parametro_Øbarras = label_Øbarras.cget("text")
10498     parametro_Asb1 = label_barra_flexion.cget("text")
10499     parametro_Asb2 = label_barra_flexion1.cget("text")
10500     parametro_A = label_A_1.cget("text")
10501     parametro_C = label_C_1.cget("text")
10502     parametro_D = label_D_1.cget("text")
10503     parametro_n = label_n_1.cget("text")
10504     parametro_dp = label_dp_1.cget("text")
10505     parametro_base_req = label_base_req.cget("text")
10506     parametro_base_req1 = label_base_req1.cget("text")
10507     parametro_check1 = label_check_As.cget("text")
10508     parametro_Mcr = label_Mcr_1.cget("text")
```

```

10509 parametro_check_Mcr = label_Mcr_check.cget("text")
10510 parametro_Rn_Mcr = label_Rn_mcr.cget("text")
10511 parametro_Rn_Mcr_check = label_Rn_mcr_check.cget("text")
10512 parametro_preq_mcr= label_preq_mcr.cget("text")
10513 parametro_As_c = label_area_req_mcr.cget("text")
10514 parametro_Ab_c = label_area_barra_mcr.cget("text")
10515 parametro_num_barra_mcr = label_num_barra_mcr.cget("text")
10516 parametro_conclusion_mcr = label_check_Mcr.cget("text")
10517 parametro_d_t = label diametro_traccion.cget("text")
10518 parametro_Ld = label_Ld_traccion.cget("text")
10519 parametro_d_c = label diametro_compresion.cget("text")
10520 parametro_Ldc = label_Ldc_compresion.cget("text")
10521 parametro_Ldh_t = label_ldh_t.cget("text")
10522 parametro_ldh_c = label_ldh_c.cget("text")
10523 parametro_ganchos = label_gancho_anclaje.cget("text")
10524 parametro_gancho_traccion = label_longitud_t.cget("text")
10525 parametro_gancho_compresion = label_longitud_c.cget("text")
10526 parametro_ancho_columna = label_ancho_columna.cget("text")
10527
10528
10529 # Agregar unidad "m" al parámetro base
10530 try:
10531     parametro_base = f"{float(entry_prueba_base.get()):.2f} m" # <<base>> con
formato y unidad
10532 except ValueError:
10533     parametro_base = "N/A" # Manejo de error si el valor no es numérico
10534
10535 parametro_r = label_recubrimiento.cget("text") # <<r>>
10536 parametro_d = label_altura_efectiva.cget("text") # <<d>>
10537
10538 # Nuevos parámetros
10539 parametro_b = parametro_base # <<b>>
10540 parametro_h = parametro_altura # <<h>>
10541
10542 parametro_info_barras_mcr = label_info_barras_mcr.cget("text") # Obtener el
texto de la etiqueta
10543
10544 # Obtener el texto de label_info_b (nuevo parámetro)
10545 parametro_info_barras = label_info_b.cget("text") # <<info_barras>>
10546
10547 # Definir la ruta de la imagen del gráfico
10548 parametro_grafico = os.path.join(current_dir, 'Reportes', 'imagenes_pilot',
'grafico.png')
10549 parametro_grafico_final = os.path.join(current_dir, 'Reportes',
'imagenes_pilot', 'grafico_final.png')
10550
10551 # Cargar el documento
10552 try:
10553     doc_report = Document(doc_template_path)
10554 except Exception as e:
10555     print(f" Error al cargar el archivo: {e}")
10556     return
10557
10558 # Diccionario de reemplazos
10559 replacements = {
10560     '<<fc>>': parametro_fc,
10561     '<<Ec>>': parametro_Ec,
10562     '<<Viga>>': parametro_viga,
10563     '<<altura>>': parametro_altura,
10564     '<<base>>': parametro_base,

```

```

10565 '<<r>>': parametro_r,
10566 '<<d>>': parametro_d,
10567 '<<b>>': parametro_b,
10568 '<<h>>': parametro_h,
10569 '<<concreto>>': parametro_concreto,
10570 '<<Wpp>>': parametro_Wpp,
10571 '<<WD>>': parametro_WD,
10572 '<<WL>>': parametro_WL,
10573 '<<Wtotal>>': parametro_Wtotal,
10574 '<<Wu>>': parametro_Wu,
10575 '<<Luz>>': parametro_luz,
10576 '<<Elemento>>': parametro_tipo_viga,
10577 '<<apoyo>>': parametro_apoyos,
10578 '<<Vmax>>': parametro_Vmax,
10579 '<<Mmax>>': parametro_Mmax,
10580 '<<Mmax1>>': parametro_Mmax_2,
10581 '<<cuantia>>': parametro_cuantia_min,
10582 '<<cuantiamax>>': parametro_cuantia_max,
10583 '<<beta>>': parametro_beta1,
10584 '<<Rn>>': parametro_Rn,
10585 '<<req>>': parametro_cuantia_req,
10586 '<<Asreq>>': parametro_acero_req,
10587 '<<Rn2>>': parametro_Rn2,
10588 '<<c2>>': parametro_C2,
10589 '<<As2>>': parametro_As2,
10590 '<<fleje>>': parametro_estribo,
10591 '<<dfleje>>': parametro_diametro_estribo,
10592 '<<Afleje>>': parametro_area_estribo,
10593 '<<Mn1>>': parametro_φMn1,
10594 '<<Mn2>>': parametro_φMn2,
10595 '<<V1>>': parametro_conclusion1,
10596 '<<V2>>': parametro_conclusion2,
10597 '<<check>>': parametro_check,
10598 '<<info_barras>>': parametro_info_barras,
10599 '<<barra>>': parametro_No_barra,
10600 '<<area>>': parametro_area_barra,
10601 '<<diametro_barra>>': parametro_Øbarras,
10602 '<<Asb1>>': parametro_Asb1,
10603 '<<Asb2>>': parametro_Asb2,
10604 '<<A>>': parametro_A,
10605 '<<C>>': parametro_C,
10606 '<<D>>': parametro_D,
10607 '<<n>>': parametro_n,
10608 '<<dp>>': parametro_dp,
10609 '<<base_req>>': parametro_base_req,
10610 '<<base_req2>>': parametro_base_req1,
10611 '<<check1>>': parametro_check1,
10612 '<<Mcr>>': parametro_Mcr,
10613 '<<check2>>': parametro_check_Mcr,
10614 '<<Rn_mcr>>': parametro_Rn_Mcr,
10615 '<<check3>>': parametro_Rn_Mcr_check,
10616 '<<preq_mcr>>': parametro_preq_mcr,
10617 '<<As_c>>': parametro_As_c,
10618 '<<Ab_c>>': parametro_Ab_c,
10619 '<<B_c>>': parametro_num_barra_mcr,
10620 '<<check4>>': parametro_conclusion_mcr,
10621 '<<Øbarra>>': parametro_d_t,
10622 '<<Ld>>': parametro_Ld,
10623 '<<Øbarra_c>>': parametro_d_c,
10624 '<<Ldc>>': parametro_Ldc,

```

```

10625         '<<Ldh_t>>': parametro_ldh_t,
10626         '<<Ldh_c>>': parametro_ldh_c,
10627         '<<gancho>>': parametro_ganchos,
10628         '<<longitud_t>>': parametro_gancho_traccion,
10629         '<<longitud_c>>': parametro_gancho_compresion,
10630         '<<check5>>': parametro_ancho_columna
10631
10632     }
10633
10634     # Reemplazar texto en todo el documento, no solo en las celdas de la tabla
10635     for paragraph in doc_report.paragraphs:
10636         for key, value in replacements.items():
10637             if key in paragraph.text:
10638                 # Reemplazar el texto en el párrafo
10639                 paragraph.text = paragraph.text.replace(key, value)
10640
10641                 # Aplicar la fuente Times New Roman
10642                 for run in paragraph.runs:
10643                     run.font.name = 'Times New Roman'
10644
10645     # Ahora reemplazar los marcadores <<JD>> dentro de las celdas de la tabla con
10646     la imagen
10647     for table in doc_report.tables:
10648         for row in table.rows:
10649             for cell in row.cells:
10650                 if '<<JD>>' in cell.text:
10651                     # Limpiar el contenido de la celda que contiene <<JD>>
10652                     for paragraph in cell.paragraphs:
10653                         for run in paragraph.runs:
10654                             run.clear() # Limpiar el contenido actual de la celda
10655
10656                     # Insertar la imagen en la celda donde estaba <<JD>>
10657                     paragraph = cell.add_paragraph()
10658                     paragraph.add_run().add_picture(parametro_grafico,
10659 width=Inches(2.5)) # Insertamos la imagen en la celda
10660
10661                     # Aplicar la fuente Times New Roman en el nuevo run de imagen
10662                     for run in paragraph.runs:
10663                         run.font.name = 'Times New Roman'
10664
10665     # Comprobar si la celda contiene el marcador <<esquema>> para
10666     insertar el gráfico final
10667     if '<<esquema>>' in cell.text:
10668         # Limpiar el contenido de la celda que contiene <<esquema>>
10669         for paragraph in cell.paragraphs:
10670             for run in paragraph.runs:
10671                 run.clear() # Limpiar el contenido actual de la celda
10672
10673         # Insertar la imagen final en la celda donde estaba
10674         <<esquema>>
10675         paragraph = cell.add_paragraph()
10676         paragraph.add_run().add_picture(parametro_grafico_final,
10677 width=Inches(3.5))
10678
10679         # Aplicar la fuente Times New Roman en el nuevo run de imagen
10680         for run in paragraph.runs:
10681             run.font.name = 'Times New Roman'
10682
10683     # Reemplazar el marcador <<info_barras>> dentro de las celdas de la tabla
10684     for table in doc_report.tables:
10685         for row in table.rows:
10686             for cell in row.cells:

```

```

10680         # Comprobar si la celda contiene el marcador <<info_barras>>
10681         if '<<info_barras>>' in cell.text:
10682             # Reemplazar el marcador con el valor de parametro_info_barras
10683             cell.text = cell.text.replace('<<info_barras>>',
parametro_info_barras)
10684
10685         # Aplicar la fuente Times New Roman (opcional)
10686         for paragraph in cell.paragraphs:
10687             for run in paragraph.runs:
10688                 run.font.name = 'Times New Roman'
10689
10690     # Reemplazar el marcador <<info_barra_As>> dentro de las celdas de la tabla
10691     for table in doc_report.tables:
10692         for row in table.rows:
10693             for cell in row.cells:
10694                 # Comprobar si la celda contiene el marcador <<info_barra_As>>
10695                 if '<<info_barra_As>>' in cell.text:
10696                     # Reemplazar el marcador con el valor de parametro_info_barras_mcr
10697                     cell.text = cell.text.replace('<<info_barra_As>>',
parametro_info_barras_mcr)
10698
10699                 # Aplicar la fuente Times New Roman (opcional)
10700                 for paragraph in cell.paragraphs:
10701                     for run in paragraph.runs:
10702                         run.font.name = 'Times New Roman'
10703
10704     # Ahora reemplazar los marcadores <<Estudiante>> y <<Modelo>> dentro de las
celdas de la tabla
10705     for table in doc_report.tables:
10706         for row in table.rows:
10707             for cell in row.cells:
10708                 if '<<Estudiante>>' in cell.text:
10709                     cell.text = cell.text.replace('<<Estudiante>>',
nombre_estudiante)
10710
10711                 # Aplicar la fuente Times New Roman en las celdas
10712                 for run in cell.paragraphs[0].runs:
10713                     run.font.name = 'Times New Roman'
10714                 if '<<Modelo>>' in cell.text:
10715                     cell.text = cell.text.replace('<<Modelo>>', nombre_modelo)
10716                 # Aplicar la fuente Times New Roman en las celdas
10717                 for run in cell.paragraphs[0].runs:
10718                     run.font.name = 'Times New Roman'
10719
10720     # Cuadro de diálogo para guardar el archivo
10721     file_path = filedialog.asksaveasfilename(
10722         defaultextension='.docx',
10723         filetypes=[('Archivos de Word', '*.docx')],
10724         title="Guardar el reporte"
10725     )
10726
10727     if file_path: # Si el usuario seleccionó un archivo y la ruta no está vacía
10728         try:
10729             doc_report.save(file_path) # Guardar el archivo en la ubicación
seleccionada
10730             print("✅ Reporte generado y guardado correctamente.")
10731
10732         # Abrir el archivo generado automáticamente en Word
10733         try:
10734             # Usamos subprocess para abrir el archivo con la aplicación
predeterminada (Word)

```

```

10734         subprocess.run(["start", file_path], check=True, shell=True)
10735         print("✅ Documento abierto correctamente.")
10736     except Exception as e:
10737         print(f" Error al abrir el archivo: {e}")
10738     except Exception as e:
10739         print(f" Error al guardar el archivo: {e}")
10740 else:
10741     print(" No se seleccionó una ubicación para guardar el archivo.")
10742
10743 def generar_reporte_cortante():
10744     # Obtener los datos de la interfaz de usuario
10745     nombre_estudiante_1 = label_estudiante.cget("text") #1
10746     nombre_modelo_1 = label_modelo.cget("text") #2
10747     parametro_fc_1 = label_fc.cget("text") #3
10748     parametro_Wu_1 = label_seccion_Wu.cget("text") #4
10749     parametro_luz_1 = label_seccion_luz.cget("text") #5
10750     parametro_Vmax_1 = label_reaccion.cget("text") #6
10751     parametro_d_1 = label_altura_efectiva.cget("text") #7
10752     parametro_estribo_1 = label_estribo.cget("text") #8
10753     parametro_area_estribo_1 = label_area_estribo.cget("text") #9
10754     parametro_Vud_1 = label_Vu.cget("text") #10
10755     parametro_Vc_1 = label_phiVc.cget("text") #11
10756     parametro_Vc_medios_1 = label_phiVc_2.cget("text") #12
10757     parametro_resistencia_concreto_1 = Vs_resistencia.cget("text") #13
10758     parametro_resistencia_estribos_1 = label_check_Vs.cget("text") #14
10759     parametro_contro_V_1 = label_check_control.cget("text") #15
10760     parametro_V_Smax_1 = Smax_estribos.cget("text") #16
10761     parametro_V_Av_1 = label_Av_requerida.cget("text") #17
10762     parametro_V_ramas_1 = label_num_ramas.cget("text") #18
10763     parametro_V_Vs_1 = Vs_resistencia_req.cget("text") #19
10764     parametro_V_check_Smax = Label_check_separacion.cget("text") #20
10765     parametro_V_check_ramas = Label_check_ramas.cget("text") #21
10766     parametro_V_Vn_1 = label_phiVn.cget("text") #22
10767     parametro_V_check_Vn = label_Vu_phiVn.cget("text") #23
10768     parametro_capacidad_Vn = label_check_Vu_phiVn.cget("text") #24
10769
10770     # Diccionario de reemplazos para etiquetas en todo el documento (excepto las
celdas)
10771     replacements_general = {
10772         "<<V_luz>>": parametro_luz_1,
10773         "<<V_fc>>": parametro_fc_1,
10774         "<<V_Wu>>": parametro_Wu_1,
10775         "<<V_vmax>>": parametro_Vmax_1,
10776         "<<V_d>>": parametro_d_1,
10777         "<<V_Estribo>>": parametro_estribo_1,
10778         "<<V_area_cortante>>": parametro_area_estribo_1,
10779         "<<V_vud>>": parametro_Vud_1,
10780         "<<V_vc>>": parametro_Vc_1,
10781         "<<V_medios>>": parametro_Vc_medios_1,
10782         "<<V_resistencia_concreto>>": parametro_resistencia_concreto_1,
10783         "<<V_resistencia_estribos>>": parametro_resistencia_estribos_1,
10784         "<<V_control_concreto>>": parametro_contro_V_1,
10785         "<<V_Smax>>": parametro_V_Smax_1,
10786         "<<V_av>>": parametro_V_Av_1,
10787         "<<V_ramas>>": parametro_V_ramas_1,

```

```

10788         "<<V_vs>>": parametro_V_Vs_1,
10789         "<<V_check_Smax>>": parametro_V_check_Smax,
10790         "<<V_check_Ramas>>": parametro_V_check_ramas,
10791         "<<V_vn>>": parametro_V_Vn_1,
10792         "<<V_check_Vn>>": parametro_V_check_Vn,
10793         "<<V_capacidad_cortante>>": parametro_capacidad_Vn
10794     }
10795
10796     # Diccionario de reemplazos para etiquetas dentro de las celdas de las tablas
10797     replacements_celdas = {
10798         "<<Estudiante>>": nombre_estudiante_1,
10799         "<<Modelo>>": nombre_modelo_1
10800     }
10801
10802     # Rutas relativas de las imágenes
10803     imagen_cortante = "Reportes/imagenes_pilot/grafica_cortante.png"
10804     imagen_triangulo = "Reportes/imagenes_pilot/longitud_refuerzo.png"
10805     imagen_despiece = "Reportes/imagenes_pilot/grafico_estribos.png"
10806
10807     # Ruta de la plantilla de Word
10808     template_path = "Reportes/Formato_cortante.docx"
10809
10810     # Abrir el documento de plantilla
10811     doc = Document(template_path)
10812
10813     # Reemplazar las etiquetas dentro de las celdas de las tablas (solo para
10814     <<Estudiante>> y <<Modelo>>)
10815     for table in doc.tables:
10816         for row in table.rows:
10817             for cell in row.cells:
10818                 # Reemplazar etiquetas de texto como <<Estudiante>> y <<Modelo>>
10819                 dentro de las celdas
10820                 for key, value in replacements_celdas.items():
10821                     if key in cell.text:
10822                         cell.text = cell.text.replace(key, value)
10823
10824                 # Reemplazar la etiqueta de imagen y eliminarla (<<V_cortante>>)
10825                 if "<<V_cortante>>" in cell.text:
10826                     cell.text = cell.text.replace("<<V_cortante>>", "") #
10827     Eliminar la etiqueta
10828     para = cell.paragraphs[0] # Tomamos el primer párrafo de la
10829     celda
10830     run = para.add_run()
10831     run.add_picture(imagen_cortante, width=Inches(3.92),
10832     height=Inches(3.45)) # Ajusta el tamaño de la imagen según lo requerido
10833
10834     # Reemplazar la etiqueta de imagen y eliminarla (<<V_triangulo>>)
10835     if "<<V_triangulo>>" in cell.text:
10836         cell.text = cell.text.replace("<<V_triangulo>>", "") #
10837     Eliminar la etiqueta
10838     para = cell.paragraphs[0] # Tomamos el primer párrafo de la
10839     celda
10840     run = para.add_run()
10841     run.add_picture(imagen_triangulo, width=Inches(4.76),
10842     height=Inches(3.12)) # Ajusta el tamaño de la imagen según lo requerido
10843
10844     # Reemplazar la etiqueta de imagen y eliminarla (<<V_despiece>>)
10845     if "<<V_despiece>>" in cell.text:
10846         cell.text = cell.text.replace("<<V_despiece>>", "") #
10847     Eliminar la etiqueta

```

```
10839         para = cell.paragraphs[0] # Tomamos el primer párrafo de la
celda
10840         run = para.add_run()
10841         run.add_picture(imagen_despiece, width=Inches(6.37),
height=Inches(2.67)) # Ajusta el tamaño de la imagen según lo requerido
10842
10843     # Reemplazar las etiquetas en el resto del documento (fuera de las celdas de
las tablas)
10844     for para in doc.paragraphs:
10845         for key, value in replacements_general.items():
10846             if key in para.text:
10847                 para.text = para.text.replace(key, value)
10848
10849     # Abrir un cuadro de diálogo para guardar el archivo
10850     if check_reporte_word_1.get() == 1: # Verificar si el Checkbutton está
seleccionado
10851         file_path = filedialog.asksaveasfilename(defaultextension=".docx",
filetypes=[("Word Documents", "*.docx")])
10852         if file_path:
10853             # Guardar el documento con el nombre proporcionado por el usuario
10854             doc.save(file_path)
10855             print(f"Reporte guardado en: {file_path}")
10856
10857             # Abrir el documento Word generado después de guardarlo
10858             if os.name == 'nt': # Para sistemas Windows
10859                 subprocess.Popen(['start', file_path], shell=True)
10860             elif os.name == 'posix': # Para sistemas Unix (Linux/Mac)
10861                 subprocess.Popen(['open', file_path])
10862             else:
10863                 print("No se puede abrir el archivo en este sistema operativo.")
10864         else:
10865             print("El archivo no fue guardado.")
10866     else:
10867         print("El reporte no se generará.")
10868
10869 # Variable para controlar si la ventana está abierta o no
10870 ventana_abierta_pilot = False
10871 ventana_Pilot = None # Para hacer la ventana accesible en ambas funciones
10872
10873 def Abrir_ventana_Pilot_Structural():
10874     global ventana_abierta_pilot, ventana_Pilot # Usamos las variables globales
para mantener el estado de la ventana
10875
10876     # Solo abrir la ventana si no está abierta ya
10877     if not ventana_abierta_pilot:
10878         # Crear una nueva ventana de tipo Toplevel
10879         ventana_Pilot = tk.Toplevel()
10880
10881         # Configurar la ventana para que no se pueda redimensionar
10882         ventana_Pilot.resizable(False, False)
10883
10884         # Asegurar que la ventana esté siempre en el frente
10885         ventana_Pilot.attributes('-topmost', True)
10886
10887         # Establecer un tamaño para la ventana
10888         ventana_Pilot.geometry('850x580')
10889
10890         # Agregar un título a la ventana
10891         ventana_Pilot.title("Acerca de Pilot Structural")
10892
```

```
10893 # Marcar que la ventana está abierta
10894 ventana_abierta_pilot = True
10895
10896 # crear contenido
10897 labelframe_pilot = tk.LabelFrame(ventana_Pilot,text="")
10898 labelframe_pilot.place(x=10,y=10,width=830,height=550)
10899
10900 labelframe_pilot_1 = tk.LabelFrame(labelframe_pilot,text="")
10901 labelframe_pilot_1.place(x=20,y=20,width=400,height=510)
10902
10903 # Crear el Notebook
10904 Notebook_pilot =
ttk.Notebook(labelframe_pilot_1,style="CustomNotebook.TNotebook")
10905
10906 # Crear una pestaña dentro del Notebook
10907 tab_pilot = ttk.Frame(Notebook_pilot)
10908
10909 # Agregar la pestaña al Notebook
10910 Notebook_pilot.add(tab_pilot, text="Información")
10911
10912 # Empaquetar el Notebook en el LabelFrame
10913 Notebook_pilot.pack(fill="both", expand=True)
10914
10915 # crear canvas
10916 canvas_pilot = tk.Canvas(tab_pilot,bg="white",bd=2,relief="flat")
10917 canvas_pilot.pack(fill="both", expand=True)
10918
10919 # Crear el Scrollbar vertical
10920 scrollbar_pilot = tk.Scrollbar(tab_pilot,
orient="vertical",command=canvas_pilot.yview)
scrollbar_pilot.place(x=370, y=0, width=20, height=475)
10921
10922 # Vincular el Scrollbar vertical al Canvas
10923 canvas_pilot.config(yscrollcommand=scrollbar_pilot.set)
10924
10925 frame_logo_pilot = tk.Frame(canvas_pilot,bd=2,relief="flat")
10926
10927 # Cargar la imagen con PIL (Pillow) y redimensionarla
10928 image_path_pilot = "Iconos_proyecto/Pilot_Structural.png"
10929 img_piloto = Image.open(image_path_pilot)
10930
10931 # Redimensionar la imagen
10932 img_piloto = img_piloto.resize((250, 250), Image.Resampling.LANCZOS)
10933
10934 # Convertir la imagen redimensionada a PhotoImage
10935 img_piloto_tk = ImageTk.PhotoImage(img_piloto)
10936
10937 # Crear un Label con la imagen redimensionada
10938 label_with_image_pilot = tk.Label(frame_logo_pilot, image=img_piloto_tk)
10939 label_with_image_pilot.place(relx=0.5, rely=0.5, anchor="center")
10940
10941 # Es necesario mantener una referencia a la imagen
10942 label_with_image_pilot.image = img_piloto_tk
10943
10944 # Crear un Label con todo el texto que mencionaste
10945 label_info_pilot = tk.Label(
10946 canvas_pilot,
10947 text="""Pilot Structural es una herramienta innovadora en el ámbito de la
ingeniería civil, diseñada para revolucionar la enseñanza y comprensión de los
conceptos técnicos del diseño de vigas de concreto reforzado. Este software
```

pionero ofrece una experiencia educativa única al integrar programación en Python, lo que permite a estudiantes explorar y entender el diseño de vigas de una manera más didáctica y accesible.

10949

10950 Con un enfoque práctico y educativo, Pilot Structural no solo facilita la comprensión de conceptos clave en el diseño estructural, sino que también automatiza la creación de reportes detallados y precisos, proporcionando análisis completos y cálculos de vigas de concreto reforzado. A través de la integración de datos dinámicos, gráficas interactivas y cálculos precisos, los usuarios pueden generar informes de forma rápida y eficiente.

10951

10952 Además, Pilot Structural destaca por su interfaz amigable, que permite a los usuarios, tanto estudiantes como profesionales, trabajar de manera intuitiva y sin complicaciones. Con este software, el aprendizaje y la práctica del diseño de vigas de concreto reforzado se vuelve más accesible, eficiente y, sobre todo, más comprensible."""

10953

```
wraplength=310,
```

10954

```
justify="left",
```

10955

```
bg="white",
```

10956

```
anchor="w"
```

10957

```
)
```

10958

```
#Crear label limitaciones
```

10959

```
label_info_pilot_1 = tk.Label(
```

10960

```
canvas_pilot,
```

10961

```
text="Limitaciones:",
```

10962

```
font=("Arial", 8, "bold"),
```

10963

```
justify="left",
```

10964

```
bg="white",
```

10965

```
anchor="w"
```

10966

```
)
```

10967

```
#Crear label limitaciones
```

10968

```
label_info_pilot_2 = tk.Label(
```

10969

```
canvas_pilot,
```

10970

```
text="""- El software solo puede realizar el análisis estructural de vigas
en una única luz, sin considerar luces continuas o estructuras con más de una luz.
```

10971

10972

10973

10974

```
- El software se especializa exclusivamente en el diseño de vigas de concreto
```

```
reforzado, por lo que no puede ser utilizado para el diseño de otros tipos de
```

```
elementos estructurales como columnas, losas, o cimentaciones.
```

10975

10976

```
- Pilot Structural se limita al diseño de vigas de concreto reforzado para flexión
```

```
y cortante. No realiza análisis o diseño para otros efectos estructurales""",
```

10977

```
wraplength=310,
```

10978

```
justify="left",
```

10979

```
bg="white",
```

10980

```
anchor="w"
```

10981

```
)
```

10982

```
labelframe_desarrollado = tk.LabelFrame(labelframe_pilot,text="")
```

10983

```
labelframe_desarrollado.place(x=430,y=20,width=370,height=510)
```

10984

```
# Crear el Notebook
```

10985

```
Notebook_pilot_1 =
```

10986

```
ttk.Notebook(labelframe_desarrollado,style="CustomNotebook.TNotebook")
```

10987

```
# Crear una pestaña dentro del Notebook
```

10988

```
tab_pilot_1 = ttk.Frame(Notebook_pilot_1)
```

10989

10990

10991

```

10992     # Agregar la pestaña al Notebook
10993     Notebook_pilot_1.add(tab_pilot_1, text="Desarrollado por:")
10994
10995     # Empaquetar el Notebook en el LabelFrame
10996     Notebook_pilot_1.pack(fill="both", expand=True)
10997
10998     # crear canvas
10999     canvas_pilot_1 = tk.Canvas(tab_pilot_1,bg="white",bd=2,relief="flat")
11000     canvas_pilot_1.pack(fill="both", expand=True)
11001
11002     # Crear el Scrollbar vertical
11003     scrollbar_pilot_1 = tk.Scrollbar(tab_pilot_1,
11004 orient="vertical",command=canvas_pilot_1.yview)
11005     scrollbar_pilot_1.place(x=340, y=0, width=20, height=475)
11006
11007     frame_desarrollador =
11008 tk.Frame(canvas_pilot_1,bd=2,relief="flat",bg="white")
11009
11010     # Cargar la imagen (ajustando la ruta relativa)
11011     ruta_desarrollo = "Iconos_proyecto/yo_ing_jd.png"
11012     imagen_desarrollo = Image.open(ruta_desarrollo)
11013     imagen_desarrollo = imagen_desarrollo.resize((250, 250))
11014     imagen_tk_desarrollo = ImageTk.PhotoImage(imagen_desarrollo)
11015
11016     # Crear un Label para mostrar la imagen dentro del Frame
11017     label_imagen_desarrollo = tk.Label(frame_desarrollador,
11018 image=imagen_tk_desarrollo, bg="white")
11019     label_imagen_desarrollo.image = imagen_tk_desarrollo
11020     label_imagen_desarrollo.pack()
11021
11022     label_desarrollador = tk.Label(canvas_pilot_1,text="Juan David Garzón
11023 Obando",font=("Arial", 12, "bold"),bg="white",
11024 justify="left",
11025 anchor="w"
11026 )
11027
11028     label_intro = tk.Label(canvas_pilot_1,text="""Soy Ingeniero civil, mi
11029 pasion por la programación se refleja en mi habilidad para desarrollar programas
11030 especializado en el área de la ingeniería estructural. Pilot Structural es un
11031 proyecto que nace de la necesidad de crear una herramienta educativa y práctica
11032 para el diseño de vigas de concreto reforzado, con el objetivo de facilitar el
11033 aprendizaje y comprensión de los conceptos estructurales.
11034 La innovación tecnológica me motiva a seguir aprendiendo y dando lo mejor de mí en
11035 cada proyecto. Me capacito de manera continua para mantenerme al día en la parte
11036 teórico-práctica y en el uso de softwares de modelado, análisis y diseño
11037 estructural, así como en la programación con Python para optimizar procesos y
11038 mejorar la eficiencia en el cálculo estructural.""",
11039 wraplength=310,
11040 justify="left",
11041 bg="white",
11042 anchor="w"
11043 )
11044
11045     label_contacto = tk.Label(canvas_pilot_1,text="Conocimiento en: Python,
11046 SAP2000, ETABS, AutoCAD, Revit, CypeCAD, Safe, Robot Structural, Microsoft Excel,
11047 Ofimatica.\n"
11048
11049 "Contacto: +57 302 543 2074\n"
11050 "Correo: Ingenieriamaximo1@gmail.com",
11051 font=("Arial", 8, "bold"),
11052 bg="white",

```

```

11037         wraplength=310,
11038         justify="left",
11039         anchor="w"
11040     )
11041
11042     label_frase_pilot = tk.Label(canvas_pilot_1, text="“El código perfecto no
existe, pero quien persiste lo mejora”",
11043         justify="center",
11044         bg="white",
11045         font=("Arial", 8, "italic"),
11046         anchor="w"
11047     )
11048
11049     labelframe_asesores_pilot = tk.LabelFrame(canvas_pilot_1, text="Asesores
del proyecto",
11050         bg="white")
11051
11052     label_asesores_pilot = tk.Label(labelframe_asesores_pilot, text="Director:
ING. Juan David Tole.\n"
11053         "Correo: juan-tole@upc.edu.co\n"
11054         "Codirectora: ING. Janneth Patricia Gil.\n"
11055         "Correo: janneth-gil@unipiloto.edu.co",
11056         justify="left",
11057         bg="white",
11058         font=("Arial", 8, "bold"),
11059         wraplength=310,
11060         anchor="w"
11061     )
11062     label_asesores_pilot.place(x=10, y=10)
11063
11064     # Vincular el Scrollbar vertical al Canvas
11065     canvas_pilot_1.config(yscrollcommand=scrollbar_pilot_1.set)
11066
11067     canvas_pilot.create_window((185, 140), window=frame_logo_pilot, width=250,
height=250)
11068     canvas_pilot.create_window((170, 450), window=label_info_pilot)
11069     canvas_pilot.create_window((55, 670), window=label_info_pilot_1)
11070     canvas_pilot.create_window((170, 780), window=label_info_pilot_2)
11071     canvas_pilot_1.create_window((170, 140), window=frame_desarrollador,
width=250, height=250)
11072     canvas_pilot_1.create_window((170, 250), window=label_desarrollador)
11073     canvas_pilot_1.create_window((170, 380), window=label_intro)
11074     canvas_pilot_1.create_window((170, 540), window=label_contacto)
11075     canvas_pilot_1.create_window((170, 590), window=label_frase_pilot)
11076     canvas_pilot_1.create_window((170, 660),
window=labelframe_asesores_pilot, width=300, height=100)
11077
11078     # Función para actualizar el área de desplazamiento del canvas
11079     def update_scrollregion(event=None):
11080         bbox = canvas_pilot.bbox("all") # Obtener las coordenadas de todo el
contenido
11081         if bbox:
11082             # Ajustar el área de desplazamiento añadiendo márgenes
11083             canvas_pilot.config(scrollregion=(0, 0, bbox[2] + 50, bbox[3] +
10))
11084
11085     # Llamar a update_scrollregion para ajustar el área de desplazamiento
11086     canvas_pilot.bind("<Configure>", update_scrollregion)
11087
11088     # Función para actualizar el área de desplazamiento del canvas

```

```
11089         def update_scrollregion(event=None):
11090             bbox = canvas_pilot_1.bbox("all") # Obtener las coordenadas de todo
el contenido
11091             if bbox:
11092                 # Ajustar el área de desplazamiento añadiendo márgenes
11093                 canvas_pilot_1.config(scrollregion=(0, 0, bbox[2] + 50, bbox[3] +
11094 10))
11095             # Llamar a update_scrollregion para ajustar el área de desplazamiento
11096             canvas_pilot_1.bind("<Configure>", update_scrollregion)
11097
11098             # Asignar la función de cerrar la ventana cuando se cierre
11099             ventana_Pilot.protocol("WM_DELETE_WINDOW", cerrar_ventana_pilot)
11100         else:
11101             pass
11102
11103     def cerrar_ventana_pilot():
11104         global ventana_abierta_pilot
11105         ventana_abierta_pilot = False
11106         ventana_Pilot.destroy()
11107
11108     ventana_abierta_structural = False
11109     ventana_structural = None
11110
11111     def Abrir_ventana_Detalle_Structural():
11112         global ventana_abierta_structural,
11113         ventana_structural, canvas_panel_2, entry_base, entry_altura, canvas_panel_3
11114
11115         # Solo abrir la ventana si no está abierta ya
11116         if not ventana_abierta_structural:
11117             # Crear una nueva ventana de tipo Toplevel
11118             ventana_structural = tk.Toplevel()
11119
11120             # Configurar la ventana para que no se pueda redimensionar
11121             ventana_structural.resizable(False, False)
11122
11123             # Asegurar que la ventana esté siempre en el frente
11124             ventana_structural.attributes('-topmost', True)
11125
11126             # Establecer un tamaño para la ventana
11127             ventana_structural.geometry('900x550')
11128
11129             # Agregar un título a la ventana
11130             ventana_structural.title("Panel modelado 3D")
11131
11132             # Crear contenido
11133             labelframe_panel_opciones = tk.LabelFrame(ventana_structural, text="")
11134             labelframe_panel_opciones.place(x=20, y=10, width=860, height=490)
11135
11136             labelframe_panel_opciones_1 =
tk.LabelFrame(labelframe_panel_opciones, text="")
11137             labelframe_panel_opciones_1.place(x=20, y=20, width=350, height=80)
11138
11139             # Crear el Notebook
11140             Notebook_panel =
ttk.Notebook(labelframe_panel_opciones_1, style="CustomNotebook.TNotebook")
11141
11142             # Crear una pestaña dentro del Notebook
11143             tab_panel = ttk.Frame(Notebook_panel)
```

```
11144
11145     # Agregar la pestaña al Notebook
11146     Notebook_panel.add(tab_panel, text="Panel de opciones")
11147
11148     # Empaquetar el Notebook en el LabelFrame
11149     Notebook_panel.pack(fill="both", expand=True)
11150
11151     # Función para redimensionar la imagen
11152     def redimensionar_imagen(ruta_imagen, tamaño):
11153     # Abrir la imagen
11154         imagen = Image.open(ruta_imagen)
11155     # Redimensionar la imagen
11156         imagen = imagen.resize(tamaño)
11157     # Convertirla a un formato que Tkinter pueda usar
11158         return ImageTk.PhotoImage(imagen)
11159
11160     # Cargar las imágenes de los iconos redimensionadas
11161     icon_1 = redimensionar_imagen("Iconos_proyecto/Flexion_VG.png", (35, 35))
11162     icon_2 = redimensionar_imagen("Iconos_proyecto/Armado_viga.png", (35, 35))
11163     icon_3 = redimensionar_imagen("Iconos_proyecto/3D_panel.png", (35, 35))
11164
11165
11166     boton_seccion_panel_1 =
11167     ttk.Button(tab_panel, image=icon_1, command=dibujar_seccion_transversal_viga)
11168     boton_seccion_panel_1.place(x=10, y=1, width=45, height=45)
11169
11170     boton_seccion_panel_2 =
11171     ttk.Button(tab_panel, image=icon_2, command=dibujar_refuerzo_viga)
11172     boton_seccion_panel_2.place(x=80, y=1, width=45, height=45)
11173
11174     boton_seccion_panel_3 =
11175     ttk.Button(tab_panel, image=icon_3, command=Dibujar_Modelo_3D)
11176     boton_seccion_panel_3.place(x=150, y=1, width=45, height=45)
11177
11178
11179     labelframe_panel_opciones_2 =
11180     tk.LabelFrame(labelframe_panel_opciones, text="")
11181     labelframe_panel_opciones_2.place(x=20, y=115, width=350, height=350)
11182
11183     # Crear el Notebook
11184     Notebook_panel_2 =
11185     ttk.Notebook(labelframe_panel_opciones_2, style="CustomNotebook.TNotebook")
11186
11187     # Crear una pestaña dentro del Notebook
11188     tab_panel_2 = ttk.Frame(Notebook_panel_2)
11189
11190     # Agregar la pestaña al Notebook
11191     Notebook_panel_2.add(tab_panel_2, text="Sección transversal")
11192
11193     # Empaquetar el Notebook en el LabelFrame
11194     Notebook_panel_2.pack(fill="both", expand=True)
11195
11196     canvas_panel_2 = tk.Canvas(tab_panel_2, bd=2, relief="flat", bg="white")
11197     canvas_panel_2.pack(fill="both", expand=True)
11198
11199     labelframe_panel_opciones_3 =
11200     tk.LabelFrame(labelframe_panel_opciones, text="")
```

```

11198         labelframe_panel_opciones_3.place(x=380,y=20 ,width=455,height=445)
11199
11200         # Crear el Notebook
11201         Notebook_panel_3 =
11202         ttk.Notebook(labelframe_panel_opciones_3,style="CustomNotebook.TNotebook")
11203
11204         # Crear una pestaña dentro del Notebook
11205         tab_panel_3 = ttk.Frame(Notebook_panel_3)
11206
11207         # Agregar la pestaña al Notebook
11208         Notebook_panel_3.add(tab_panel_3, text="Modelo 3D")
11209
11210         # Empaquetar el Notebook en el LabelFrame
11211         Notebook_panel_3.pack(fill="both", expand=True)
11212
11213         canvas_panel_3 = tk.Canvas(tab_panel_3,bg="white")
11214         canvas_panel_3.pack(fill="both", expand=True)
11215
11216         #Crear botones ventana
11217         boton_aceptar_pilot =
11218         ttk.Button(ventana_structural,text="Aceptar",command=guardar_imagen_pilot)
11219         boton_aceptar_pilot.place(x=300,y=510,width=100)
11220
11221         boton_cancelar_pilot =
11222         ttk.Button(ventana_structural,text="Cancelar",command=cerrar_ventana_1)
11223         boton_cancelar_pilot.place(x=500,y=510,width=100)
11224
11225         # Marcar que la ventana está abierta
11226         ventana_abierta_structural = True
11227
11228         # Cuando la ventana se cierre, marcar que no está abierta
11229         ventana_structural.protocol("WM_DELETE_WINDOW", cerrar_ventana_1)
11230
11231         def cerrar_ventana_1():
11232             global ventana_abierta_structural
11233             ventana_abierta_structural = False
11234             ventana_structural.destroy()
11235
11236         # Variable global para controlar si la ventana está abierta o no
11237         ventana_abierta_3D = False
11238         ventana_3D = None # Variable para la ventana Toplevel
11239
11240         # Función para abrir la ventana 3D
11241         def Dibujar_Modelo_3D():
11242             global ventana_3D,
11243             ventana_abierta_3D,entry_ancho_3D,entry_altura_3D,entry_recubrimiento_3D,entry_Luz
11244             _3D,canvas_panel_3
11245             global
11246             combo_estribos,combo_num_estribo,entry_n_estribo,entry_tramos_separacion,treeview_
11247             estribos,combo_n_barra,entry_diametro_b
11248             global
11249             combo_n_barra_sup,entry_diametro_b_sup,entry_barra_n,entry_barra_n_sup,check_drawb
11250             ar_var
11251
11252             if not ventana_abierta_3D:
11253                 ventana_3D = tk.Toplevel()
11254                 ventana_3D.title("Modelo 3D")
11255                 ventana_3D.geometry('610x430')
11256                 ventana_3D.resizable(False, False)
11257                 ventana_3D.attributes('-topmost', True)

```

```

11249
11250 labelframe_3D = tk.LabelFrame(ventana_3D,text="")
11251 labelframe_3D.place(x=20,y=20,width=570,height=360)
11252
11253 frame_3D_2 = tk.Frame(labelframe_3D,bd=2,relief="raised")
11254 frame_3D_2.place(x=30,y=25,width=510,height=300)
11255
11256 # Crear el Notebook
11257 Notebook_3D = ttk.Notebook(frame_3D_2,style="CustomNotebook.TNotebook")
11258
11259 # Crear una pestaña dentro del Notebook
11260 tab_3D = ttk.Frame(Notebook_3D)
11261 tab_3D_1 = ttk.Frame(Notebook_3D)
11262 tab_3D_2 = ttk.Frame(Notebook_3D)
11263
11264 # Agregar la pestaña al Notebook
11265 Notebook_3D.add(tab_3D, text="Geometría")
11266 Notebook_3D.add(tab_3D_1, text="Estribos")
11267 Notebook_3D.add(tab_3D_2, text="Barras - Longitudinal")
11268
11269 # Empaquetar el Notebook en el LabelFrame
11270 Notebook_3D.pack(fill="both", expand=True)
11271
11272 # Agregar contenido
11273 labelframe_geometria = tk.LabelFrame(tab_3D, text="Geometría")
11274 labelframe_geometria.place(x=10,y=1,width=485,height=260)
11275
11276 check_geometria_var = tk.BooleanVar()
11277 check_geometria_3D = tk.Checkbutton(labelframe_geometria, text="Ver
parámetros geométricos", variable=check_geometria_var,
command=Ver_parametros_viga)
11278 check_geometria_3D.place(x=10,y=10)
11279
11280
11281 labelframe_parametros_3D =
tk.LabelFrame(labelframe_geometria,text="Parámetros geométricos [m]")
11282 labelframe_parametros_3D.place(x=10,y=40,width=200,height=190)
11283
11284 label_ancho_3D = tk.Label(labelframe_parametros_3D,text="Ancho [b]:")
11285 label_ancho_3D.place(x=10,y=10)
11286
11287 entry_ancho_3D = tk.Entry(labelframe_parametros_3D)
11288 entry_ancho_3D.place(x=80,y=10,width=100)
11289
11290 label_altura_3D = tk.Label(labelframe_parametros_3D,text="Altura [h]:")
11291 label_altura_3D.place(x=10,y=50)
11292
11293 entry_altura_3D = tk.Entry(labelframe_parametros_3D)
11294 entry_altura_3D.place(x=80,y=50,width=100)
11295
11296 label_recubrimiento_3D = tk.Label(labelframe_parametros_3D,text="Capa
[r]:")
11297 label_recubrimiento_3D.place(x=10,y=90)
11298
11299 entry_recubrimiento_3D = tk.Entry(labelframe_parametros_3D)
11300 entry_recubrimiento_3D.place(x=80,y=90,width=100)
11301
11302 label_Luz_3D = tk.Label(labelframe_parametros_3D,text="Longitud [L]:")
11303 label_Luz_3D.place(x=5,y=130)
11304
11305 entry_Luz_3D = tk.Entry(labelframe_parametros_3D)

```

```

11306     entry_Luz_3D.place(x=80,y=130,width=100)
11307
11308     frame_img_3D =
tk.Frame(labelframe_geometria,bd=3,relief="ridge",bg="white")
11309     frame_img_3D.place(x=240,y=46,width=200,height=184)
11310
11311     # Cargar la imagen usando Pillow
11312     Image_viga_3D = "Iconos_proyecto/esquema_3D.png"
11313     img_viga_3D = Image.open(Image_viga_3D)
11314     img_viga_3D = img_viga_3D.resize((200, 200))
11315     img_viga_3D = ImageTk.PhotoImage(img_viga_3D)
11316
11317     # Mostrar la imagen en el frame
11318     label_img_3D = Label(frame_img_3D, image=img_viga_3D)
11319     label_img_3D.image = img_viga_3D
11320     label_img_3D.pack(fill="both", expand=True)
11321
11322     check_modelo_var = tk.BooleanVar()
11323     check_modelo_3D = tk.Checkbutton(labelframe_geometria, text="Dibujar
modelo paramétrico", variable=check_modelo_var,
11324                                     command=actualizar_viga)
11325     check_modelo_3D.place(x=240,y=10)
11326
11327     canvas_estribos_3D = tk.Canvas(tab_3D_1)
11328     canvas_estribos_3D.pack(fill="both", expand=True)
11329
11330     canvas_barras_3D = tk.Canvas(tab_3D_2)
11331     canvas_barras_3D.pack(fill="both", expand=True)
11332
11333     #Crear el Scrollbar vertical
11334     scrollbar_barras = tk.Scrollbar(tab_3D_2,
orient="vertical",command=canvas_barras_3D.yview)
11335     scrollbar_barras.place(x=480, y=0, width=20, height=265)
11336
11337     #Crear el Scrollbar vertical
11338     scrollbar_3D = tk.Scrollbar(tab_3D_1,
orient="vertical",command=canvas_estribos_3D.yview)
11339     scrollbar_3D.place(x=480, y=0, width=20, height=265)
11340
11341     # Vincular el Scrollbar vertical al Canvas
11342     canvas_barras_3D.config(yscrollcommand=scrollbar_barras.set)
11343
11344     # Vincular el Scrollbar vertical al Canvas
11345     canvas_estribos_3D.config(yscrollcommand=scrollbar_3D.set)
11346
11347     labelframe_estribos_3D = tk.LabelFrame(canvas_estribos_3D,text="Estribos")
11348
11349     check_estribo_var = tk.BooleanVar()
11350     checkbutton_estribo_5 =
tk.Checkbutton(labelframe_estribos_3D,text="Distancia del primer estribo /
apoyo",variable=check_estribo_var,
11351
)
11352     checkbutton_estribo_5.place(x=10,y=10)
11353
11354     entry_estribo_5 = tk.Entry(labelframe_estribos_3D)
11355     entry_estribo_5.place(x=270,y=15,width=100)
11356     entry_estribo_5.insert(0, "0.05")
11357     entry_estribo_5.config(state="disabled")
11358
11359     labelframe_tramos = tk.LabelFrame(labelframe_estribos_3D,text="Tramos")

```

```

11360     labelframe_tramos.place(x=20,y=40,width=415,height=80)
11361
11362     label_tramos = tk.Label(labelframe_tramos,text="#Tramos:")
11363     label_tramos.place(x=10,y=15)
11364
11365     # Crear el Combobox con la lista de datos
11366     combo_estribos = ttk.Combobox(labelframe_tramos, values=[1, 2, 3])
11367     combo_estribos.place(x=70,y=15,width=85)
11368     combo_estribos.set("Seleccionar")
11369
11370     Label_tramos_separacion = tk.Label(labelframe_tramos,text="Separación:")
11371     Label_tramos_separacion.place(x=180,y=15)
11372
11373     entry_tramos_separacion = tk.Entry(labelframe_tramos)
11374     entry_tramos_separacion.place(x=260,y=15,width=85)
11375
11376     labelframe_secciones =
tk.LabelFrame(labelframe_estribos_3D,text="Distribución de estribos por secciones
de tramos")
11377     labelframe_secciones.place(x=20,y=130,width=415,height=220)
11378
11379     # Ruta de la imagen
11380     ruta_imagen_3D = "Iconos_proyecto/check_estribos.png"
11381
11382     # Cargar la imagen con Pillow
11383     imagen_original_3D = Image.open(ruta_imagen_3D)
11384
11385     # Redimensionar la imagen para que ocupe todo el tamaño del botón (25x25)
11386     imagen_redimensionada_3D = imagen_original_3D.resize((16, 16),
Image.Resampling.LANCZOS)
11387
11388     # Convertir la imagen a PhotoImage para usarla en Tkinter
11389     imagen_fig_3D = ImageTk.PhotoImage(imagen_redimensionada_3D)
11390
11391     # Crear el botón y asignar la imagen
11392     Button_verificar = ttk.Button(labelframe_secciones,
image=imagen_fig_3D,command=distribución_estribos)
11393     Button_verificar.place(x=360, y=8, width=25, height=25)
11394
11395     # Mantener la referencia a la imagen para evitar que se elimine
11396     Button_verificar.image = imagen_fig_3D
11397
11398     label_num_estribo = tk.Label(labelframe_secciones,text="Estribo:")
11399     label_num_estribo.place(x=10,y=10)
11400
11401     combo_num_estribo = ttk.Combobox(labelframe_secciones, values=
["#2", "#3", "#4"])
11402     combo_num_estribo.place(x=70,y=10,width=85)
11403     combo_num_estribo.set("Seleccionar")
11404
11405     label_n_estribo = tk.Label(labelframe_secciones,text="#Estribos [n]:")
11406     label_n_estribo.place(x=180,y=10)
11407
11408     entry_n_estribo = tk.Entry(labelframe_secciones)
11409     entry_n_estribo.place(x=260,y=10,width=85)
11410
11411     # Crear el primer Treeview
11412     treeview_estribos = ttk.Treeview(labelframe_secciones,columns=
("Secciones","Tramo 1", "Tramo 2", "Tramo 3"), show="headings")
11413     treeview_estribos.place(x=20,y=50,width=355,height=100)

```

```

11414
11415     # Definir los encabezados de las columnas
11416     treeview_estribos.heading("Secciones", text="Secciones")
11417     treeview_estribos.heading("Tramo 1", text="Tramo 1")
11418     treeview_estribos.heading("Tramo 2", text="Tramo 2")
11419     treeview_estribos.heading("Tramo 3", text="Tramo 3")
11420
11421     # Modificar el ancho de las columnas
11422     treeview_estribos.column("Secciones", width=87, stretch=False,
anchor="center")
11423     treeview_estribos.column("Tramo 1", width=87, stretch=False,
anchor="center")
11424     treeview_estribos.column("Tramo 2", width=87, stretch=False,
anchor="center")
11425     treeview_estribos.column("Tramo 3", width=87, stretch=False,
anchor="center")
11426
11427     # Insertar filas en el Treeview
11428     treeview_estribos.insert("", "end", values=("#Estribos", "", "", ""))
11429     treeview_estribos.insert("", "end", values=("#Ramas", "", "", ""))
11430     treeview_estribos.insert("", "end", values=("Separación (m)", "", "", ""))
11431
11432     # Función para restaurar el tamaño de las columnas después de intentar
redimensionar
11433     def no_resize(event):
11434         treeview_estribos.column("Secciones", width=87, stretch=False,
anchor="center")
11435         treeview_estribos.column("Tramo 1", width=87, stretch=False,
anchor="center")
11436         treeview_estribos.column("Tramo 2", width=87, stretch=False,
anchor="center")
11437         treeview_estribos.column("Tramo 3", width=87, stretch=False,
anchor="center")
11438
11439     # Vincular el evento de ButtonRelease-1 para evitar el redimensionamiento
11440     treeview_estribos.bind("<ButtonRelease-1>", no_resize)
11441
11442     check_draw_var = tk.BooleanVar()
11443     checkbutton_draw_estribos =
tk.Checkbutton(labelframe_secciones, text="Dibujar confinamiento
transversal", variable=check_draw_var,
11444                 command=dibujar_confinamiento_estribos)
11445     checkbutton_draw_estribos.place(x=10, y=170)
11446
11447     labelframe_barras_long = tk.LabelFrame(tab_3D_2, text="Barras -
Longitudinal")
11448     labelframe_ref_inferior =
tk.LabelFrame(labelframe_barras_long, text="Refuerzo Inferior")
11449     labelframe_ref_inferior.place(x=10, y=10, width=435, height=200)
11450     label_n_barra = tk.Label(labelframe_ref_inferior, text="#Barra:")
11451     label_n_barra.place(x=10, y=10)
11452     combo_n_barra = ttk.Combobox(labelframe_ref_inferior, values=
["#2", "#3", "#4", "#5", "#6", "#7", "#8", "#9", "#10", "#11", "#14", "#18"])
11453     combo_n_barra.place(x=70, y=10, width=85)
11454     combo_n_barra.set("Seleccionar")
11455     combo_n_barra.bind("<<ComboboxSelected>>", actualizacion_diametro_barras)
11456     label_material_barra = tk.Label(labelframe_ref_inferior, text="Material:")
11457     label_material_barra.place(x=180, y=10)
11458     combo_material = ttk.Combobox(labelframe_ref_inferior, values=["420 Mpa"])
11459     combo_material.place(x=240, y=10, width=85)

```

```

11460     combo_material.set("Seleccionar")
11461     label_barras_n = tk.Label(labelframe_ref_inferior,text="#Barras [n]:")
11462     label_barras_n.place(x=5,y=50)
11463     entry_barra_n = tk.Entry(labelframe_ref_inferior)
11464     entry_barra_n.place(x=70,y=50,width=85)
11465     label_diametro_b = tk.Label(labelframe_ref_inferior,text="ØBarra:")
11466     label_diametro_b.place(x=180,y=50)
11467     entry_diametro_b = tk.Entry(labelframe_ref_inferior)
11468     entry_diametro_b.place(x=240,y=50,width=85)
11469
11470     #Colocar barras inferior
11471     var_ext_bar = tk.IntVar()
11472     radiobutton_ext_bar =
tk.Radiobutton(labelframe_ref_inferior,text="Extender barras inferior
principal",variable=var_ext_bar,value=1)
11473     radiobutton_ext_bar.place(x=10,y=90)
11474     #Colocar ganchos de 90°
11475     var_gancho_bar = tk.IntVar()
11476     radiobutton_gancho_inf =
tk.Radiobutton(labelframe_ref_inferior,text="Colocar gancho de
90°",variable=var_gancho_bar,value=1)
11477     radiobutton_gancho_inf.place(x=10,y=130)
11478
11479     labelframe_ref_superior =
tk.LabelFrame(labelframe_barras_long,text="Refuerzo Superior")
11480     labelframe_ref_superior.place(x=10,y=220,width=435,height=200)
11481     label_n_barra_sup = tk.Label(labelframe_ref_superior,text="#Barra:")
11482     label_n_barra_sup.place(x=10,y=10)
11483     combo_n_barra_sup = ttk.Combobox(labelframe_ref_superior, values=
["#2", "#3", "#4", "#5", "#6", "#7", "#8", "#9", "#10", "#11", "#14", "#18"])
11484     combo_n_barra_sup.place(x=70,y=10,width=85)
11485     combo_n_barra_sup.set("Seleccionar")
11486     combo_n_barra_sup.bind("<<ComboboxSelected>>",
actualizacion_diametro_barras)
11487     label_material_barra_sup =
tk.Label(labelframe_ref_superior,text="Material:")
11488     label_material_barra_sup.place(x=180,y=10)
11489     combo_material_sup = ttk.Combobox(labelframe_ref_superior, values=["420
Mpa"])
11490     combo_material_sup.place(x=240,y=10,width=85)
11491     combo_material_sup.set("Seleccionar")
11492     label_barras_n_sup = tk.Label(labelframe_ref_superior,text="#Barras [n]:")
11493     label_barras_n_sup.place(x=5,y=50)
11494     entry_barra_n_sup = tk.Entry(labelframe_ref_superior)
11495     entry_barra_n_sup.place(x=70,y=50,width=85)
11496     label_diametro_b_sup = tk.Label(labelframe_ref_superior,text="ØBarra:")
11497     label_diametro_b_sup.place(x=180,y=50)
11498     entry_diametro_b_sup = tk.Entry(labelframe_ref_superior)
11499     entry_diametro_b_sup.place(x=240,y=50,width=85)
11500
11501     #Colocar barras superior
11502     var_ext_bar_sup = tk.IntVar()
11503     radiobutton_ext_bar_sup =
tk.Radiobutton(labelframe_ref_superior,text="Extender barras superior
principal",variable=var_ext_bar_sup,value=1)
11504     radiobutton_ext_bar_sup.place(x=10,y=90)
11505     #Colocar ganchos de 90° superior
11506     var_gancho_bar_sup = tk.IntVar()
11507     radiobutton_gancho_sup =
tk.Radiobutton(labelframe_ref_superior,text="Colocar gancho de

```

```

90°",variable=var_gancho_bar_sup,value=1)
11508     radiobutton_gancho_sup.place(x=10,y=130)
11509
11510     check_drawbar_var = tk.BooleanVar()
11511     checkbutton_estribo_5 =
tk.Checkbutton(labelframe_barras_long,text="Dibujar barras
longitudinal",variable=check_drawbar_var,
11512                 command=Dibujar_barras_longitudinal)
11513     checkbutton_estribo_5.place(x=10,y=435)
11514
11515     entry_barra_n.bind("<KeyRelease>", actualizar_entradas_barras)
11516     entry_barra_n_sup.bind("<KeyRelease>", actualizar_entradas_barras)
11517
11518     boton_3D_cancelar =
ttk.Button(ventana_3D,text="Cancelar",command=cerrar_ventana_3D)
11519     boton_3D_cancelar.place(x=240,y=390,width=100)
11520
11521     canvas_estribos_3D.create_window((240,210),
window=labelframe_estribos_3D,width=460,height=400)
11522     canvas_barras_3D.create_window((240,260),
window=labelframe_barras_long,width=460,height=500)
11523
11524     # Cuando se cierra la ventana, reiniciar la variable de control
11525     ventana_3D.protocol("WM_DELETE_WINDOW", cerrar_ventana_3D) # Captura del
evento de cierre de ventana
11526
11527     # Función para actualizar el área de desplazamiento del canvas
11528     def update_scrollregion(event=None):
11529         bbox = canvas_barras_3D.bbox("all") # Obtener las coordenadas de todo
el contenido
11530         if bbox:
11531             # Ajustar el área de desplazamiento añadiendo márgenes
11532             canvas_barras_3D.config(scrollregion=(0, 0, bbox[2] + 50, bbox[3]
+ 10))
11533
11534     # Llamar a update_scrollregion para ajustar el área de desplazamiento
11535     canvas_barras_3D.bind("<Configure>", update_scrollregion)
11536
11537
11538     # Función para actualizar el área de desplazamiento del canvas
11539     def update_scrollregion(event=None):
11540         bbox = canvas_estribos_3D.bbox("all") # Obtener las coordenadas de
todo el contenido
11541         if bbox:
11542             # Ajustar el área de desplazamiento añadiendo márgenes
11543             canvas_estribos_3D.config(scrollregion=(0, 0, bbox[2] + 50,
bbox[3] + 10))
11544
11545     # Llamar a update_scrollregion para ajustar el área de desplazamiento
11546     canvas_estribos_3D.bind("<Configure>", update_scrollregion)
11547
11548     # Marcar la ventana como abierta
11549     ventana_abierta_3D = True
11550
11551     # Función para cerrar la ventana 3D
11552     def cerrar_ventana_3D():
11553         global ventana_3D, ventana_abierta_3D
11554         if ventana_3D:
11555             ventana_3D.destroy()
11556             ventana_abierta_3D = False

```

```

11557     ventana_3D = None
11558
11559 def actualizacion_diametro_barras(event):
11560     # Mapeo de las barras y sus diámetros en milímetros
11561     diametros_barra = {
11562         "#2": 6.4,
11563         "#3": 9.5,
11564         "#4": 12.7,
11565         "#5": 15.9,
11566         "#6": 19.1,
11567         "#7": 22.2,
11568         "#8": 25.4,
11569         "#9": 28.7,
11570         "#10": 32.3,
11571         "#11": 35.8,
11572         "#14": 43.0,
11573         "#18": 57.3
11574     }
11575
11576     # Obtener el valor seleccionado del ComboBox
11577     seleccion_barra = combo_n_barra.get()
11578     seleccion_superior = combo_n_barra_sup.get()
11579
11580     # Actualizar el valor del Entry con el diámetro correspondiente
11581     if seleccion_barra in diametros_barra:
11582         entry_diametro_b.delete(0, tk.END) # Limpiar el Entry antes de actualizar
11583         entry_diametro_b.insert(0, f"{diametros_barra[seleccion_barra]} mm")
11584     # Actualizar el valor del Entry correspondiente al ComboBox superior
11585     if seleccion_superior in diametros_barra:
11586         entry_diametro_b_sup.delete(0, tk.END) # Limpiar el Entry antes de
actualizar
11587         entry_diametro_b_sup.insert(0, f"{diametros_barra[seleccion_superior]}
mm")
11588
11589 def actualizar_entradas_barras(*args):
11590     # Obtener los datos de los Entry
11591     valor_inferior = entry_barra_n.get() # Valor de entry_barra_n
11592     valor_superior = entry_barra_n_sup.get() # Valor de entry_barra_n_sup
11593
11594     # Actualizar los valores en los Entry correspondientes
11595     entry_barras_inferior.delete(0, tk.END) # Limpiar el Entry antes de
actualizar
11596     entry_barras_inferior.insert(0, valor_inferior) # Insertar el valor de
entry_barra_n
11597
11598     entry_barras_superior.delete(0, tk.END) # Limpiar el Entry antes de
actualizar
11599     entry_barras_superior.insert(0, valor_superior) # Insertar el valor de
entry_barra_n_sup
11600
11601 def distribución_estribos():
11602     # Obtener valores de los widgets
11603     opcion_dis_1 = combo_estribos.get()
11604     tipo_estribo_dis = combo_num_estribo.get()
11605     num_estribo_dis_1 = entry_n_estribo.get()
11606     separacion_dis_1 = entry_tramos_separacion.get()
11607
11608     if not (opcion_dis_1 and tipo_estribo_dis and num_estribo_dis_1 and
separacion_dis_1):
11609         return # No hacer nada si hay campos vacíos

```

```
11610
11611     opcion_dis_1 = int(opcion_dis_1)
11612     numero_ramas_E = 2 # Valor fijo de número de ramas
11613
11614     # Obtener los identificadores de las filas existentes
11615     filas = treeview_estribos.get_children()
11616     valores_actuales = {fila: list(treeview_estribos.item(fila, "values")) for
fila in filas}
11617
11618     # Actualizar solo la columna seleccionada
11619     columna = opcion_dis_1
11620     valores_actuales[filas[0]][columna] = f"
{num_estribo_dis_1}E{tipo_estribo_dis}"
11621     valores_actuales[filas[1]][columna] = numero_ramas_E
11622     valores_actuales[filas[2]][columna] = separacion_dis_1
11623
11624     for fila, valores in valores_actuales.items():
11625         treeview_estribos.item(fila, values=valores)
11626
11627     # ACTUALIZAR LOS ENTRY EN labelframe_reporte
11628     if opcion_dis_1 == 1:
11629         entry_tramo_1.delete(0, tk.END)
11630         entry_tramo_1.insert(0, num_estribo_dis_1)
11631         entry_tramo_1_S.delete(0, tk.END)
11632         entry_tramo_1_S.insert(0, separacion_dis_1)
11633
11634     elif opcion_dis_1 == 2:
11635         entry_tramo_2.delete(0, tk.END)
11636         entry_tramo_2.insert(0, num_estribo_dis_1)
11637         entry_tramo_2_S.delete(0, tk.END)
11638         entry_tramo_2_S.insert(0, separacion_dis_1)
11639
11640     elif opcion_dis_1 == 3:
11641         entry_tramo_3.delete(0, tk.END)
11642         entry_tramo_3.insert(0, num_estribo_dis_1)
11643         entry_tramo_3_S.delete(0, tk.END)
11644         entry_tramo_3_S.insert(0, separacion_dis_1)
11645
11646
11647 # Función para obtener los datos y actualizar los entrys
11648 def Ver_parametros_viga():
11649     # Obtener los valores de los Entry del labelframe_reporte
11650     base_parametro = entry_prueba_seccion_base.get()
11651     altura_parametro = entry_prueba_seccion_altura.get()
11652     recubrimiento_parametro = entry_prueba_seccion_recubrimiento.get()
11653     longitud_parametro = entry_prueba_longitud.get()
11654
11655     # Actualizar los valores de los Entry correspondientes en la ventana 3D
11656     entry_ancho_3D.delete(0, tk.END) # Limpiar el Entry antes de poner el nuevo
valor
11657     entry_ancho_3D.insert(0, base_parametro) # Actualizar con el valor de
base_parametro
11658
11659     entry_altura_3D.delete(0, tk.END)
11660     entry_altura_3D.insert(0, altura_parametro) # Actualizar con el valor de
altura_parametro
11661
11662     entry_recubrimiento_3D.delete(0, tk.END)
11663     entry_recubrimiento_3D.insert(0, recubrimiento_parametro) # Actualizar con el
valor de recubrimiento_parametro
```

```

11664
11665     entry_Luz_3D.delete(0, tk.END)
11666     entry_Luz_3D.insert(0, longitud_parametro) # Actualizar con el valor de
longitud_parametro
11667
11668
11669 # Variables globales para almacenar la figura y el canvas
11670 fig = None
11671 ax = None
11672 canvas = None
11673
11674 def dibujar_viga(base_3D, altura_3D, longitud_3D, canvas_panel_3):
11675     """
11676     Dibuja una viga con las dimensiones especificadas y muestra la figura en el
11677     canvas de Tkinter.
11678     """
11679     global fig, ax, canvas
11680     # Crear la figura y el eje 3D
11681     fig = plt.figure()
11682     ax = fig.add_subplot(111, projection='3d')
11683
11684     color_lineas_poligono = 'grey'
11685
11686     # Función para dibujar un polígono con sombreado
11687     def dibujar_poligono(vertices, color='grey', color_lineas='grey'):
11688         poligono = Poly3DCollection([vertices], color=color,
11689         edgecolor=color_lineas, alpha=0.3)
11690         ax.add_collection3d(poligono)
11691
11692     # Definir las caras de la viga
11693     offset_x = base_3D / 2
11694     caras = [
11695         [[-offset_x, 0, 0], [offset_x, 0, 0], [offset_x, longitud_3D, 0], [-
11696         offset_x, longitud_3D, 0]],
11697         [[-offset_x, 0, altura_3D], [offset_x, 0, altura_3D], [offset_x,
11698         longitud_3D, altura_3D], [-offset_x, longitud_3D, altura_3D]],
11699         [[-offset_x, 0, 0], [-offset_x, 0, altura_3D], [-offset_x, longitud_3D,
11700         altura_3D], [-offset_x, longitud_3D, 0]],
11701         [[offset_x, 0, 0], [offset_x, 0, altura_3D], [offset_x, longitud_3D,
11702         altura_3D], [offset_x, longitud_3D, 0]]
11703     ]
11704     for cara in caras:
11705         dibujar_poligono(cara)
11706
11707     ax.set_xlabel(f'Base ({base_3D} m)')
11708     ax.set_ylabel(f'Longitud ({longitud_3D} m)')
11709     ax.set_zlabel(f'Altura ({altura_3D} m)')
11710
11711     # Ajustar la proporción de la caja y los límites de los ejes
11712     ax.set_box_aspect([1, longitud_3D / base_3D, altura_3D / base_3D])
11713     ax.set_xlim([-offset_x, offset_x])
11714     ax.set_ylim([0, longitud_3D])
11715     ax.set_zlim([0, altura_3D])
11716
11717     ax.set_xticks([])
11718     ax.set_yticks([])
11719     ax.set_zticks([])
11720
11721     # Mostrar la figura en el Canvas de Tkinter usando FigureCanvasTkAgg
11722     canvas = FigureCanvasTkAgg(fig, master=canvas_panel_3)

```

```

11717 canvas.get_tk_widget().pack(fill="both", expand=True)
11718 canvas.draw()
11719
11720 # Cerrar la figura después de renderizarla para liberar memoria
11721 plt.close(fig)
11722
11723 # Función para obtener los valores de los Entry
11724 def obtener_valores_entry():
11725     global longitud_3D, altura_3D, base_3D, tramo1, tramo2, tramo3,
desfase_estribo
11726     # Aquí se pueden obtener los valores de los Tkinter Entry, por ejemplo:
11727     longitud_3D = float(entry_prueba_longitud.get()) # Obtiene el valor de
longitud
11728     altura_3D = float(entry_prueba_seccion_altura.get()) # Obtiene el valor de
altura
11729     base_3D = float(entry_prueba_seccion_base.get()) # Obtiene el valor de base
11730     tramo1 = (int(entry_tramo_1.get()), float(entry_tramo_1_S.get())) # Obtiene
los valores del tramo 1
11731     tramo2 = (int(entry_tramo_2.get()), float(entry_tramo_2_S.get())) # Obtiene
los valores del tramo 2
11732     tramo3 = (int(entry_tramo_3.get()), float(entry_tramo_3_S.get())) # Obtiene
los valores del tramo 3
11733
11734     # Captura el valor del desfase (recubrimiento) desde el Entry
11735     desfase_estribo = float(entry_prueba_seccion_recubrimiento.get())
11736
11737 # Lista para almacenar las líneas de los estribos
11738 estribos_lines = []
11739
11740 # Función para dibujar una línea 3D (esto se usa para dibujar los estribos)
11741 def dibujar_linea(x, y, z, color='black'):
11742     line = ax.plot3D(x, y, z, color=color)
11743     return line
11744
11745 # Función para dibujar estribos correctamente en ambas caras y almacenar las
líneas
11746 def dibujar_estribos_ref(inicio, n_estribos, separacion, color, cara, offset_x):
11747     global estribos_lines
11748
11749     for i in range(n_estribos):
11750         posicion = inicio + i * separacion
11751         if posicion > longitud_3D:
11752             break # Evita dibujar fuera de la viga
11753
11754         # Dibujar y almacenar las líneas de estribos
11755         if cara == 'izquierda':
11756             line1 = dibujar_linea([-offset_x + desfase_estribo, offset_x -
desfase_estribo], [posicion, posicion], [desfase_estribo, desfase_estribo],
color=color)
11757             line2 = dibujar_linea([-offset_x + desfase_estribo, offset_x -
desfase_estribo], [posicion, posicion], [altura_3D - desfase_estribo, altura_3D -
desfase_estribo], color=color)
11758             line3 = dibujar_linea([-offset_x + desfase_estribo, -offset_x +
desfase_estribo], [posicion, posicion], [desfase_estribo, altura_3D -
desfase_estribo], color=color)
11759             line4 = dibujar_linea([offset_x - desfase_estribo, offset_x -
desfase_estribo], [posicion, posicion], [desfase_estribo, altura_3D -
desfase_estribo], color=color)
11760         elif cara == 'derecha':

```

```

11761         line1 = dibujar_linea([-offset_x + desfase_estribo, offset_x -
desfase_estribo], [longitud_3D - posicion, longitud_3D - posicion],
[desfase_estribo, desfase_estribo], color=color)
11762         line2 = dibujar_linea([-offset_x + desfase_estribo, offset_x -
desfase_estribo], [longitud_3D - posicion, longitud_3D - posicion], [altura_3D -
desfase_estribo, altura_3D - desfase_estribo], color=color)
11763         line3 = dibujar_linea([-offset_x + desfase_estribo, -offset_x +
desfase_estribo], [longitud_3D - posicion, longitud_3D - posicion],
[desfase_estribo, altura_3D - desfase_estribo], color=color)
11764         line4 = dibujar_linea([offset_x - desfase_estribo, offset_x -
desfase_estribo], [longitud_3D - posicion, longitud_3D - posicion],
[desfase_estribo, altura_3D - desfase_estribo], color=color)
11765
11766         # Almacenar las líneas en la lista
11767         estribos_lines.extend([line1, line2, line3, line4])
11768
11769 # Función para limpiar los estribos previos
11770 def limpiar_estribos():
11771     global estribos_lines
11772     for line in estribos_lines:
11773         line[0].remove() # Elimina cada línea almacenada
11774     estribos_lines = [] # Vaciar la lista de líneas
11775
11776 # Función para dibujar los estribos con las correcciones en ambas caras y
actualizar el gráfico
11777 def dibujar_confinamiento_estribos():
11778     """
11779     Dibuja el confinamiento transversal dentro del mismo gráfico de la viga,
usando los estribos.
11780     """
11781     global ax, canvas
11782
11783     if ax is None or canvas is None:
11784         return # No hay gráfico para modificar
11785
11786     # Limpiar los estribos previos sin borrar la figura 3D
11787     limpiar_estribos()
11788
11789     # Llamamos a la función para obtener los valores actualizados de los Entry
11790     obtener_valores_entry()
11791
11792     # Calcular inicios correctos de los tramos sin superposición
11793     inicio_tramo1 = 0.05
11794     inicio_tramo2 = inicio_tramo1 + tramo1[0] * tramo1[1]
11795     mitad_viga = longitud_3D / 2
11796     posiciones_tramo3 = np.linspace(mitad_viga - (tramo3[0] - 1) / 2 * tramo3[1],
mitad_viga + (tramo3[0] - 1) / 2 * tramo3[1],
11797                                     tramo3[0])
11798
11799     # **Asegúrate de pasar el offset_x en cada llamada**
11800     offset_x = base_3D / 2 # Calcula el valor de offset_x aquí
11801
11802     # Dibujar los estribos con las correcciones en ambas caras
11803     dibujar_estribos_ref(inicio_tramo1, tramo1[0], tramo1[1], color='red',
cara='izquierda', offset_x=offset_x)
11804     dibujar_estribos_ref(inicio_tramo2, tramo2[0], tramo2[1], color='darkgreen',
cara='izquierda', offset_x=offset_x)
11805     dibujar_estribos_ref(inicio_tramo1, tramo1[0], tramo1[1], color='red',
cara='derecha', offset_x=offset_x)

```

```
11807     dibujar_estribos_ref(inicio_tramo2, tramo2[0], tramo2[1], color='darkgreen',
11808     cara='derecha', offset_x=offset_x)
11809     # Dibujar estribos para el tramo 3
11810     for pos in posiciones_tramo3:
11811         dibujar_estribos_ref(pos, 1, 0, color='blue', cara='izquierda',
11812         offset_x=offset_x)
11813         dibujar_estribos_ref(pos, 1, 0, color='blue', cara='derecha',
11814         offset_x=offset_x)
11815     # Actualizar la figura
11816     canvas.draw()
11817 # Lista para almacenar las líneas de las barras longitudinales
11818 barras_lines = []
11819 ganchos_lines = []
11820
11821 def dibujar_linea(x, y, z, color='blue', grosor=1):
11822     """Dibuja una línea en el gráfico 3D"""
11823     line = ax.plot3D(x, y, z, color=color, linewidth=grosor)
11824     return line
11825
11826 def dibujar_gancho(x, y, z, color='grey', direccion='abajo'):
11827     """Dibuja un gancho en un nodo específico y lo retorna para poder eliminarlo
11828     después."""
11829     longitud_gancho = 0.15
11830
11831     if direccion == 'abajo':
11832         line = ax.plot([x, x], [y, y], [z, z - longitud_gancho], color=color,
11833         linewidth=2)
11834     elif direccion == 'arriba':
11835         line = ax.plot([x, x], [y, y], [z, z + longitud_gancho], color=color,
11836         linewidth=2)
11837
11838     return line
11839
11840 def Dibujar_barras_longitudinal():
11841     """
11842     Dibuja las barras longitudinales en el gráfico 3D
11843     cuando se activa el checkbox correspondiente.
11844     """
11845     global ax, canvas, barras_lines, ganchos_lines
11846
11847     if ax is None or canvas is None:
11848         return # Evita errores si no hay un gráfico activo
11849
11850     # Verifica si el Checkbutton está activo
11851     if check_drawbar_var.get() == 0:
11852         limpiar_barras()
11853         return
11854
11855     # Obtener valores de entrada
11856     longitud_3D = float(entry_prueba_longitud.get())
11857     altura_3D = float(entry_prueba_seccion_altura.get())
11858     base_3D = float(entry_prueba_seccion_base.get())
11859     desfase_estribo = float(entry_prueba_seccion_recubrimiento.get())
11860
11861     n_barras_sup = int(entry_barras_superior.get()) # Número de barras superiores
11862     n_barras_inf = int(entry_barras_inferior.get()) # Número de barras inferiores
```

```

11861
11862     # Parámetros de separación
11863     limite_x_sup_inf = (base_3D / 2) - desfase_estribo
11864     limite_z_inf = desfase_estribo
11865     ajuste_sup = desfase_estribo * 0.8 # Ajuste para subir las barras superiores
11866     limite_z_sup = altura_3D - desfase_estribo - desfase_estribo + ajuste_sup #
11867     Se sube un poco más
11868     # Calcular posiciones de las barras superiores e inferiores
11869     posiciones_barras_x_sup = np.linspace(-limite_x_sup_inf, limite_x_sup_inf,
11870     n_barras_sup)
11871     posiciones_barras_x_inf = np.linspace(-limite_x_sup_inf, limite_x_sup_inf,
11872     n_barras_inf)
11873     # Dibujar barras superiores y ganchos
11874     for i in range(n_barras_sup):
11875         x_sup = [posiciones_barras_x_sup[i], posiciones_barras_x_sup[i]]
11876         y = [0, longitud_3D] # Extensión a lo largo de la viga
11877         z_sup = [limite_z_sup, limite_z_sup] # Se suben un poco más
11878         barras_lines.append(dibujar_linea(x_sup, y, z_sup, color='black',
11879         grosor=2))
11880     # Ganchos en ambas caras
11881     ganchos_lines.append(dibujar_gancho(posiciones_barras_x_sup[i], 0,
11882     limite_z_sup, color='black', direccion='abajo'))
11883     ganchos_lines.append(dibujar_gancho(posiciones_barras_x_sup[i],
11884     longitud_3D, limite_z_sup, color='black', direccion='abajo'))
11885     # Dibujar barras inferiores (sin cambios)
11886     for i in range(n_barras_inf):
11887         x_inf = [posiciones_barras_x_inf[i], posiciones_barras_x_inf[i]]
11888         y = [0, longitud_3D]
11889         z_inf = [limite_z_inf, limite_z_inf]
11890         barras_lines.append(dibujar_linea(x_inf, y, z_inf, color='darkred',
11891         grosor=2))
11892     # Ganchos en ambas caras
11893     ganchos_lines.append(dibujar_gancho(posiciones_barras_x_inf[i], 0,
11894     limite_z_inf, color='darkred', direccion='arriba'))
11895     ganchos_lines.append(dibujar_gancho(posiciones_barras_x_inf[i],
11896     longitud_3D, limite_z_inf, color='darkred', direccion='arriba'))
11897     # Actualizar gráfico
11898     canvas.draw()
11899 def limpiar_barras():
11900     """Elimina del gráfico las barras longitudinales y sus ganchos"""
11901     global barras_lines, ganchos_lines
11902     for line in barras_lines:
11903         line[0].remove()
11904     for gancho in ganchos_lines:
11905         gancho[0].remove()
11906     barras_lines = []
11907     ganchos_lines = []
11908     canvas.draw()
11909

```

```

11911
11912 # Función para actualizar la viga con los nuevos datos
11913 def actualizar_viga():
11914     try:
11915         # Obtener los valores de los campos de entrada
11916         base_3D = float(entry_ancho_3D.get())
11917         altura_3D = float(entry_altura_3D.get())
11918         longitud_3D = float(entry_Luz_3D.get())
11919
11920         # Limpiar el canvas antes de dibujar la nueva viga
11921         for widget in canvas_panel_3.winfo_children():
11922             widget.destroy()
11923
11924         # Llamar a la función para dibujar la viga en el canvas
11925         dibujar_viga(base_3D, altura_3D, longitud_3D, canvas_panel_3)
11926
11927     except ValueError:
11928         print("Error: Ingresa valores numéricos válidos")
11929
11930
11931
11932 # Variable de control para saber si la ventana está abierta
11933 ventana_abierta_dimension = False
11934 ventana_dimension = None # Esta será la referencia a la ventana, inicializada
11935                             como None
11936 fig = None
11937 canvas_fig_rect = None
11938
11938 # Función para dibujar la sección transversal de la viga
11939 def dibujar_seccion_transversal_viga():
11940     global ventana_abierta_dimension, ventana_dimension,
11941     entry_altura, entry_r, entry_d, canvas_panel_2, entry_base, fig,
11942     canvas_fig_rect, entry_l
11943
11942     if ventana_abierta_dimension: # Si la ventana ya está abierta, no se abre
11943         otra
11944         return
11945
11945     # Crear una ventana de Toplevel
11946     ventana_dimension = tk.Toplevel()
11947     ventana_dimension.geometry("300x300")
11948     ventana_dimension.title("Dimensiones de la viga")
11949     ventana_dimension.resizable(False, False)
11950     ventana_dimension.attributes('-topmost', True)
11951
11952     # Crear contenido
11953     labelframe_dimension_viga = tk.LabelFrame(ventana_dimension, text="Dimensiones
11954 [m]")
11955     labelframe_dimension_viga.place(x=15, y=15, width=270, height=240)
11956
11956     label_base = tk.Label(labelframe_dimension_viga, text="Ancho:")
11957     label_base.place(x=30, y=10)
11958
11959     entry_base = tk.Entry(labelframe_dimension_viga)
11960     entry_base.place(x=100, y=10, width=100)
11961
11962     label_altura = tk.Label(labelframe_dimension_viga, text="Altura:")
11963     label_altura.place(x=30, y=50)
11964
11965     entry_altura = tk.Entry(labelframe_dimension_viga)

```

```
11966     entry_altura.place(x=100,y=50,width=100)
11967
11968     label_r_viga= tk.Label(labelframe_dimension_viga,text="recubrimiento:")
11969     label_r_viga.place(x=10,y=90)
11970
11971     entry_r = tk.Entry(labelframe_dimension_viga)
11972     entry_r.place(x=100,y=90,width=100)
11973
11974     label_d= tk.Label(labelframe_dimension_viga,text="Altura util:")
11975     label_d.place(x=10,y=130)
11976
11977     entry_d = tk.Entry(labelframe_dimension_viga)
11978     entry_d.place(x=100,y=130,width=100)
11979
11980     label_l = tk.Label(labelframe_dimension_viga,text="Longitud:")
11981     label_l.place(x=10,y=170)
11982
11983     entry_l = tk.Entry(labelframe_dimension_viga)
11984     entry_l.place(x=100,y=170,width=100)
11985
11986     boton_dibujar_seccion =
11987     ttk.Button(ventana_dimension,text="Dibujar",command=dibujar_rectangulo_seccion)
11988     boton_dibujar_seccion.place(x=30,y=260,width=100)
11989
11989     boton_cancelar_seccion =
11990     ttk.Button(ventana_dimension,text="Cancelar",command=cerrar_ventana_viga)
11991     boton_cancelar_seccion.place(x=170,y=260,width=100)
11992
11993     # Cambiar la variable de control para indicar que la ventana está abierta
11994     ventana_abierta_dimension = True
11995
11996     # Evento para cerrar la ventana
11997     ventana_dimension.protocol("WM_DELETE_WINDOW", cerrar_ventana_viga)
11998
11999     # Vincular las entradas de altura y recubrimiento al cálculo de la altura
12000     efectiva_util
12001     entry_altura.bind("<KeyRelease>", calcular_altura_efectiva_util) # Cuando se
12002     presiona una tecla en entry_altura
12003     entry_r.bind("<KeyRelease>", calcular_altura_efectiva_util) # Cuando se
12004     presiona una tecla en entry_r
12005
12006     # Función para cuando se cierre la ventana
12007     def cerrar_ventana_viga():
12008         global ventana_abierta_dimension, ventana_dimension
12009         ventana_abierta_dimension = False
12010         if ventana_dimension:
12011             ventana_dimension.destroy()
12012
12013     # Función para calcular la altura efectiva util
12014     def calcular_altura_efectiva_util(*args):
12015         try:
12016             # Obtener los valores de altura y recubrimiento
12017             altura_util = float(entry_altura.get())
12018             recubrimiento_util = float(entry_r.get())
12019
12020             # Calcular la altura efectiva util
12021             altura_efectiva_util = altura_util - recubrimiento_util
```

```

12021         # Actualizar el resultado en entry_d
12022         entry_d.delete(0, tk.END) # Borrar el valor anterior
12023         entry_d.insert(0, f"{altura_efectiva_util:.2f}") # Insertar el resultado
con dos decimales
12024
12025         # Actualizar el valor de recubrimiento_util en
entry_prueba_seccion_recubrimiento
12026         entry_prueba_seccion_recubrimiento.delete(0, tk.END) # Borrar el valor
anterior
12027         entry_prueba_seccion_recubrimiento.insert(0, f"{recubrimiento_util:.2f}")
# Insertar el recubrimiento
12028
12029     except ValueError:
12030         # En caso de que no se ingresen números válidos, limpiar el campo d
12031         entry_d.delete(0, tk.END)
12032         entry_prueba_seccion_recubrimiento.delete(0, tk.END)
12033
12034
12035 def dibujar_rectangulo_seccion():
12036     global fig, canvas_fig_rect, combo_seccion_estribo
12037
12038     try:
12039         base_rect = float(entry_base.get())
12040         altura_rec = float(entry_altura.get())
12041
12042         # Obtener el dato de entry_l
12043         longitud_vg = entry_l.get()
12044
12045         if longitud_vg:
12046             # Actualizar el entry_prueba_longitud con el valor de entry_l
12047             entry_prueba_longitud.delete(0, tk.END) # Borrar el contenido actual
12048             entry_prueba_longitud.insert(0, longitud_vg) # Insertar el valor de
entry_l en entry_prueba_longitud
12049
12050             if fig is not None:
12051                 plt.close(fig)
12052
12053                 # Escalar dimensiones (metros a cm o una escala manejable)
12054                 escala = 100 # 1 metro = 100 unidades para mejor visualización
12055                 base_esc = base_rect * escala
12056                 altura_esc = altura_rec * escala
12057
12058                 fig, ax = plt.subplots(figsize=(5, 5))
12059
12060                 # Dibujar el rectángulo escalado
12061                 ax.add_patch(plt.Rectangle((0, 0), base_esc, altura_esc, edgecolor="none"
, facecolor="gray"))
12062
12063                 # Ajustar límites para mantener margen
12064                 ax.set_xlim(-10, base_esc + 10)
12065                 ax.set_ylim(-10, altura_esc + 10)
12066
12067                 # Mantener escala igual y ocultar ejes
12068                 ax.set_aspect('equal')
12069                 ax.axis('off')
12070
12071                 # Ajustar márgenes para evitar bordes
12072                 plt.subplots_adjust(left=0, right=1, top=1, bottom=0)
12073
12074                 # Mostrar en el canvas de Tkinter

```

```

12075         if canvas_fig_rect is not None:
12076             canvas_fig_rect.get_tk_widget().destroy()
12077
12078             canvas_fig_rect = FigureCanvasTkAgg(fig, master=canvas_panel_2)
12079             canvas_fig_rect.draw()
12080             canvas_fig_rect.get_tk_widget().pack(fill=tk.BOTH, expand=True)
12081
12082             # Actualizar el Entry con el nuevo formato
12083             entry_prueba_seccion_estribo.delete(0, tk.END) # Borrar el contenido
actual
12084             entry_prueba_seccion_estribo.insert(0, f'VG {base_rect * 100:.2f} X
{altura_rec * 100:.2f}')
12085
12086             entry_prueba_seccion_base.delete(0, tk.END) # Borrar el contenido actual
12087             entry_prueba_seccion_base.insert(0, f'{base_rect}')
12088
12089             entry_prueba_seccion_altura.delete(0, tk.END)
12090             entry_prueba_seccion_altura.insert(0, f'{altura_rec}')
12091
12092         except ValueError:
12093             pass
12094
12095         # Lista para almacenar las barras y leyendas dibujadas
12096         barras_dibujadas = []
12097         leyendas_dibujadas = []
12098
12099         def dibujar_elementos_refuerzo():
12100             global fig, canvas_fig_rect, entry_prueba_seccion_altura,
entry_prueba_seccion_base, entry_prueba_seccion_recubrimiento, entry_numero_1,
entry_numero_inferior_1, entry_fila_barra_1, entry_fila_inferior_1,
entry_prueba_seccion_estribo, barras_dibujadas, leyendas_dibujadas
12101             global entry_prueba_estribo_diseño
12102             try:
12103                 if not entry_prueba_seccion_altura.winfo_exists():
12104                     return
12105                 if not entry_prueba_seccion_base.winfo_exists():
12106                     return
12107                 if not entry_prueba_seccion_recubrimiento.winfo_exists():
12108                     return
12109                 if not entry_numero_1.winfo_exists():
12110                     return
12111                 if not entry_numero_inferior_1.winfo_exists():
12112                     return
12113                 if not entry_fila_barra_1.winfo_exists():
12114                     return
12115                 if not entry_fila_inferior_1.winfo_exists():
12116                     return
12117                 if not entry_prueba_seccion_estribo.winfo_exists():
12118                     return
12119                 if not entry_prueba_estribo_diseño.winfo_exists():
12120                     return
12121
12122                 # Obtener valores de los Entry y verificar si están vacíos
12123                 base_text = entry_prueba_seccion_base.get().strip()
12124                 altura_text = entry_prueba_seccion_altura.get().strip()
12125                 offset_text = entry_prueba_seccion_recubrimiento.get().strip()
12126                 numero_barras_text = entry_numero_1.get().strip()
12127                 numero_barras_inferior_text = entry_numero_inferior_1.get().strip()
12128                 fila_barra_1_text = entry_fila_barra_1.get().strip()
12129                 fila_inferior_1_text = entry_fila_inferior_1.get().strip()

```

```

12130     seccion_viga_text = entry_prueba_seccion_estribo.get().strip()
12131     estribo_diseño_text = entry_prueba_estribo_diseño.get().strip()
12132
12133     if base_text == "" or altura_text == "" or offset_text == "" or
numero_barras_text == "" or numero_barras_inferior_text == "" or fila_barra_1_text
== "" or fila_inferior_1_text == "" or seccion_viga_text == "" or
estribo_diseño_text == "":
12134         return
12135
12136     # Convertir a float
12137     base_rect = float(base_text) * 100 # Convertir metros a cm
12138     altura_rect = float(altura_text) * 100
12139     offset = float(offset_text) * 100 # Offset en cm
12140     numero_barras = int(numero_barras_text) # Número de barras superiores
12141     numero_barras_inferior = int(numero_barras_inferior_text) # Número de
barras inferiores
12142
12143     if fig is None:
12144         return
12145
12146     ax = fig.gca()
12147
12148     # Eliminar las barras de refuerzo anteriores
12149     while barras_dibujadas:
12150         barras_dibujadas.pop().remove()
12151
12152     # Eliminar leyendas anteriores
12153     while leyendas_dibujadas:
12154         leyenda = leyendas_dibujadas.pop()
12155         leyenda.remove()
12156
12157     # Dibujar el rectángulo interno con offset
12158     ax.add_patch(plt.Rectangle((offset, offset), base_rect - 2 * offset,
altura_rect - 2 * offset,
12159                             edgecolor="red", facecolor="none", linestyle="-",
linewidth=2))
12160
12161     # Parámetros de las barras de refuerzo
12162     radio_barra = 0.0105 * 100
12163     separacion_barra = 0.250 * 100
12164
12165     # Coordenadas de la primera y última barra superior (bordes del rectángulo
interno)
12166     x_inicio = offset + radio_barra
12167     x_final = base_rect - offset - radio_barra
12168     y_barras_superiores = altura_rect - offset - radio_barra
12169     y_barras_inferiores = offset + radio_barra
12170
12171     # Función para dibujar barras de refuerzo
12172     def dibujar_barras(numero_barras, y_posicion):
12173         if numero_barras == 1:
12174             barra = ax.add_patch(plt.Circle((base_rect / 2, y_posicion),
radius=radio_barra, color='black', label="Barra de acero"))
12175             barras_dibujadas.append(barra)
12176         elif numero_barras >= 2:
12177             x_positions = [x_inicio]
12178             if numero_barras > 2:
12179                 for i in range(1, numero_barras - 1):
12180                     x_positions.append(x_inicio + i * (x_final - x_inicio) /
(numero_barras - 1))

```

```

12181         x_positions.append(x_final)
12182
12183         for x in x_positions:
12184             barra = ax.add_patch(plt.Circle((x, y_posicion),
radius=radio_barra, color='black', label="Barra de acero"))
12185             barras_dibujadas.append(barra)
12186
12187             # Dibujar las barras superiores
12188             dibujar_barras(numero_barras, y_barras_superiores)
12189
12190             # Dibujar las barras inferiores
12191             dibujar_barras(numero_barras_inferior, y_barras_inferiores)
12192
12193             # Crear las leyendas con marcadores como puntos negros y añadir los
valores de fila_barra_1 y fila_inferior_1
12194             leyenda_superior = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='black', markersize=7, label=f'Barras: {numero_barras_text}
{fila_barra_1_text}')
12195             leyenda_inferior = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='black', markersize=7, label=f'Barras:
{numero_barras_inferior_text}{fila_inferior_1_text}')
12196
12197             # Nueva leyenda para la sección de la viga
12198             leyenda_seccion_viga = plt.Line2D([0], [0], marker='s', color='grey',
markerfacecolor='grey', markersize=8, linestyle='None', label=f'Sección:
{seccion_viga_text}')
12199             leyenda_estribo = plt.Line2D([0], [0], marker='s', color='r',
markerfacecolor='white', markersize=8, linestyle='None',
label=f'Estribo: E{estribo_diseño_text}')
12200
12201
12202
12203             # Primera leyenda para las barras
12204             leyenda_1 = ax.legend(handles=[leyenda_superior, leyenda_inferior],
fontsize=8, loc='upper center', ncol=2)
12205             ax.add_artist(leyenda_1)
12206
12207             # Segunda leyenda para la sección de la viga
12208             leyenda_2 = ax.legend(handles=[leyenda_seccion_viga, leyenda_estribo],
fontsize=8, loc='lower center', ncol=1)
12209
12210
12211             canvas_fig_rect.draw() # Actualizar el gráfico
12212
12213         except ValueError:
12214             pass
12215
12216     def guardar_imagen_pilot():
12217         ruta_carpeta = "Reportes/imagenes_pilot"
12218         os.makedirs(ruta_carpeta, exist_ok=True) # Crea la carpeta si no existe
12219
12220         ruta_imagen_pilot = os.path.join(ruta_carpeta, "grafico_pilot.png")
12221
12222         # Captura el gráfico actual de Matplotlib y guárdalo
12223         plt.savefig(ruta_imagen_pilot, dpi=300, bbox_inches='tight')
12224         print(f"Imagen guardada en: {ruta_imagen_pilot}")
12225
12226         actualizar_imagen_pilot(ruta_imagen_pilot)
12227
12228     def actualizar_imagen_pilot(ruta_imagen):
12229         # Cargar la imagen en el Label dentro del frame

```

```
12230     imagen_pilot_1 = Image.open(ruta_imagen)
12231     imagen_pilot_1 = imagen_pilot_1.resize((250, 250), Image.LANCZOS) #
Redimensionar al tamaño del frame
12232     imagenTk_1 = ImageTk.PhotoImage(imagen_pilot_1)
12233
12234     # Actualizar la imagen en el Label
12235     label_imagen.config(image=imagenTk_1)
12236     label_imagen.image = imagenTk_1 # Evita que la imagen sea eliminada por el
recolector de basura
12237
12238
12239 # Variable global para verificar si la ventana ya está abierta
12240 ventana_refuerzo_abierta_1 = False
12241
12242 def dibujar_refuerzo_viga():
12243     global
ventana_refuerzo_abierta_1,ventana_refuerzo_viga,treeview_armado,label_asignar_che
ck,combo_seccion_estribo,combo_seccion_barra,Entry_doblez
12244     global entry_A_confi , entry_C_confi, entry_B_confi, entry_D_confi,
combo_gancho_estribo, entry_G_confi, label_ok
12245
12246     # Verificar si la ventana ya está abierta
12247     if ventana_refuerzo_abierta_1:
12248         return
12249
12250     ventana_refuerzo_abierta_1 = True
12251
12252     # Crear la ventana secundaria
12253     ventana_refuerzo_viga = tk.Toplevel()
12254     ventana_refuerzo_viga.geometry("600x310")
12255     ventana_refuerzo_viga.title("Refuerzo de la viga")
12256     ventana_refuerzo_viga.resizable(False, False)
12257     ventana_refuerzo_viga.attributes('-topmost', True)
12258
12259     labelframe_armado = tk.LabelFrame(ventana_refuerzo_viga,text="")
12260     labelframe_armado.place(x=10,y=10,width=580,height=250)
12261
12262     # Crear el Notebook
12263     Notebook_armado =
ttk.Notebook(labelframe_armado,style="CustomNotebook.TNotebook")
12264
12265     # Crear una pestaña dentro del Notebook
12266     tab_armado = ttk.Frame(Notebook_armado)
12267     tab_armado_1 = ttk.Frame(Notebook_armado)
12268
12269     # Agregar la pestaña al Notebook
12270     Notebook_armado.add(tab_armado, text="Refuerzo longitudinal")
12271     Notebook_armado.add(tab_armado_1, text="Refuerzo transversal")
12272
12273     # Empaquetar el Notebook en el LabelFrame
12274     Notebook_armado.pack(fill="both", expand=True)
12275
12276     labelframe_asignar = tk.LabelFrame(tab_armado,text="Asignación de refuerzo")
12277     labelframe_asignar.place(x=10,y=10,width=460,height=150)
12278
12279     frame_asignar_check = tk.Frame(tab_armado,bd=2,relief="sunken")
12280     frame_asignar_check.place(x=10,y=165,width=460,height=50)
12281
12282     label_asignar_check = tk.Label(frame_asignar_check,text="",
12283                                     wraplength=400,
```

```

12284         justify="left",
12285         anchor="w")
12286     label_asignar_check.place(x=1,y=1)
12287
12288     boton_asignar =
12289     ttk.Button(tab_armado,text="Asignar",command=abrir_subventana_asignar)
12290     boton_asignar.place(x=470,y=100,width=100)
12291
12292     # Crear el primer Treeview
12293     treeview_armado = ttk.Treeview(labelframe_asignar,columns=("Refuerzo", "#Barra
12294     ", "Fila 1", "#Barra", "Fila 2"), show="headings")
12295     treeview_armado.place(x=10,y=15,width=440,height=100)
12296
12297     # Definir los encabezados de las columnas
12298     treeview_armado.heading("Refuerzo", text="Refuerzo")
12299     treeview_armado.heading("#Barra ", text="#Barra")
12300     treeview_armado.heading("Fila 1", text="Fila 1")
12301     treeview_armado.heading("#Barra", text="#Barra")
12302     treeview_armado.heading("Fila 2", text="Fila 2")
12303
12304     # Modificar el ancho de las columnas
12305     treeview_armado.column("Refuerzo", width=87, stretch=False, anchor="center")
12306     treeview_armado.column("#Barra ", width=87, stretch=False, anchor="center")
12307     treeview_armado.column("Fila 1", width=87, stretch=False, anchor="center")
12308     treeview_armado.column("#Barra", width=87, stretch=False, anchor="center")
12309     treeview_armado.column("Fila 2", width=87, stretch=False, anchor="center")
12310
12311     labelframe_d_estribo = tk.LabelFrame(tab_armado_1,text="Dimensionamiento
12312     estribo")
12313     labelframe_d_estribo.place(x=10,y=10,width=250,height=200)
12314
12315     frame_d_estribo = tk.Frame(labelframe_d_estribo,bd=2,relief="flat",bg="white")
12316     frame_d_estribo.pack(fill="both", expand=True)
12317
12318     # Ruta de la imagen
12319     image_path_estribo = "Iconos_proyecto/Dimension_estribo.png"
12320     # Cargar la imagen usando PIL (Pillow)
12321     img_estribo = Image.open(image_path_estribo)
12322     # Redimensionar la imagen
12323     img_estribo = img_estribo.resize((170, 170), Image.Resampling.LANCZOS)
12324     # Convertir la imagen redimensionada a PhotoImage
12325     img_estribo_tk = ImageTk.PhotoImage(img_estribo)
12326     # Crear un Label con la imagen redimensionada y colocarlo en el Frame
12327     label_with_image_estribo = tk.Label(labelframe_d_estribo,
12328     image=img_estribo_tk,bg="white")
12329     label_with_image_estribo.place(relx=0.5, rely=0.5, anchor="center")
12330     # Es necesario mantener una referencia a la imagen para evitar que sea
12331     eliminada
12332     label_with_image_estribo.image = img_estribo_tk
12333
12334     labelframe_parametros = tk.LabelFrame(tab_armado_1,text="Parametros de
12335     figurado")
12336     labelframe_parametros.place(x=270,y=10,width=290,height=200)
12337
12338     label_seccion_estribo = tk.Label(labelframe_parametros,text="Sección:")
12339     label_seccion_estribo.place(x=10,y=10)
12340
12341     combo_seccion_estribo = ttk.Combobox(labelframe_parametros)
12342     combo_seccion_estribo.place(x=60,y=10,width=88)

```

```
12338 label_seccion_barra = tk.Label(labelframe_parametros,text="#Barra:")
12339 label_seccion_barra.place(x=150,y=10)
12340
12341 combo_seccion_barra = ttk.Combobox(labelframe_parametros,values=["#3","#4"])
12342 combo_seccion_barra.place(x=200,y=10,width=80)
12343
12344 label_gancho_estribo = tk.Label(labelframe_parametros,text="Gancho:")
12345 label_gancho_estribo.place(x=10,y=35)
12346
12347 combo_gancho_estribo = ttk.Combobox(labelframe_parametros,values=["135°"])
12348 combo_gancho_estribo.place(x=60,y=35,width=88)
12349
12350 label_doblez = tk.Label(labelframe_parametros,text="Doblez:")
12351 label_doblez.place(x=150,y=35)
12352
12353 Entry_doblez = tk.Entry(labelframe_parametros)
12354 Entry_doblez.place(x=200,y=35,width=80)
12355
12356 labelframe_confinados = tk.LabelFrame(labelframe_parametros,text="Elementos
confinados [cm]")
12357 labelframe_confinados.place(x=10,y=65,width=270,height=105)
12358
12359 label_A_confi = tk.Label(labelframe_confinados,text="A:")
12360 label_A_confi.place(x=10,y=10)
12361
12362 entry_A_confi = tk.Entry(labelframe_confinados)
12363 entry_A_confi.place(x=30,y=10,width=50)
12364
12365 label_B_confi = tk.Label(labelframe_confinados,text="B:")
12366 label_B_confi.place(x=90,y=10)
12367
12368 entry_B_confi = tk.Entry(labelframe_confinados)
12369 entry_B_confi.place(x=110,y=10,width=50)
12370
12371 label_C_confi = tk.Label(labelframe_confinados,text="C:")
12372 label_C_confi.place(x=170,y=10)
12373
12374 entry_C_confi = tk.Entry(labelframe_confinados)
12375 entry_C_confi.place(x=190,y=10,width=50)
12376
12377 label_D_confi = tk.Label(labelframe_confinados,text="D:")
12378 label_D_confi.place(x=10,y=45)
12379
12380 entry_D_confi = tk.Entry(labelframe_confinados)
12381 entry_D_confi.place(x=30,y=45,width=50)
12382
12383 label_G_confi = tk.Label(labelframe_confinados,text="G:")
12384 label_G_confi.place(x=90,y=45)
12385
12386 entry_G_confi = tk.Entry(labelframe_confinados)
12387 entry_G_confi.place(x=110,y=45,width=50)
12388
12389 label_ok = tk.Label(labelframe_confinados,bd=2,relief="sunken",text="")
12390 label_ok.place(x=180,y=45,width=80)
12391
12392 boton_dibujar_refuerzo_viga =
ttk.Button(ventana_refuerzo_viga,text="Dibujar",command=dibujar_elementos_refuerzo
)
12393 boton_dibujar_refuerzo_viga.place(x=100,y=270,width=100)
12394
```

```

12395     boton_aceptar_refuerzo_viga = ttk.Button(ventana_refuerzo_viga,text="Aceptar")
12396     boton_aceptar_refuerzo_viga .place(x=250,y=270,width=100)
12397
12398     boton_cancelar_refuerzo_viga =
12399     ttk.Button(ventana_refuerzo_viga,text="Cancelar",command=al_cerrar)
12400     boton_cancelar_refuerzo_viga .place(x=400,y=270,width=100)
12401
12402     # Función para restaurar el tamaño de las columnas después de intentar
12403     redimensionar
12404     def no_resize(event):
12405         treeview_armado.column("Refuerzo", width=87, stretch=False,
12406         anchor="center")
12407         treeview_armado.column("#Barra ", width=87, stretch=False,
12408         anchor="center")
12409         treeview_armado.column("Fila 1", width=87, stretch=False, anchor="center")
12410         treeview_armado.column("#Barra", width=87, stretch=False, anchor="center")
12411         treeview_armado.column("Fila 2", width=87, stretch=False, anchor="center")
12412
12413     # Vincular el evento de ButtonRelease-1 para evitar el redimensionamiento
12414     treeview_armado.bind("<ButtonRelease-1>", no_resize)
12415
12416     # Configurar el evento de cierre para restablecer la bandera
12417     ventana_refuerzo_viga.protocol("WM_DELETE_WINDOW", al_cerrar)
12418     combo_seccion_estribo.bind('<FocusIn>', lambda event:
12419     Dimensionamiento_estribos(event, entry_prueba_seccion_estribo,
12420     combo_seccion_estribo))
12421     # Asociar la función al evento de selección del Combobox
12422     combo_seccion_barra.bind('<<ComboboxSelected>>', lambda event:
12423     Elementos_confinados(event, combo_seccion_barra, Entry_doble,
12424     entry_prueba_seccion_estribo,
12425
12426     entry_prueba_seccion_recubrimiento, entry_A_confi, entry_C_confi,entry_B_confi,
12427     entry_D_confi,entry_prueba_estribo_diseño))
12428     combo_gancho_estribo.bind('<<ComboboxSelected>>', lambda event:
12429     Extension_corte_estribo(event, combo_gancho_estribo, combo_seccion_barra,
12430     entry_G_confi, label_ok))
12431
12432     def al_cerrar():
12433         global ventana_refuerzo_abierta_1
12434         ventana_refuerzo_abierta_1 = False
12435         ventana_refuerzo_viga.destroy()
12436
12437     subventana_asignar_abierta = False
12438
12439     def abrir_subventana_asignar():
12440         global subventana_asignar_abierta,subventana_asignar,combo_agregar_barras,
12441         entry_ba_1 ,entry_ba_2,combo_agregar_refuerzo,combo_agregar_filas
12442         global entry_mas_barra_1 ,entry_mas_barra_2
12443         ,entry_cantidad_barras,entry_mas_ref_1,entry_barras_1,entry_mas_ref_2,label_info_b
12444         arras_ref
12445         global label_info_barras_inferior,treeview_armado,label_asignar_check
12446
12447         # Verificar si la subventana ya está abierta
12448         if subventana_asignar_abierta:
12449             return
12450
12451         subventana_asignar_abierta = True
12452
12453         # Crear la subventana

```

```
12440 subventana_asignar = tk.Toplevel()
12441 subventana_asignar.geometry("530x265")
12442 subventana_asignar.title("Asignar barras")
12443 subventana_asignar.resizable(False, False)
12444 subventana_asignar.attributes('-topmost', True)
12445
12446 labelframe_asignar_datos = tk.LabelFrame(subventana_asignar, text="")
12447 labelframe_asignar_datos.place(x=10, y=10, width=510, height=210)
12448
12449 label_agregar_refuerzo = tk.Label(labelframe_asignar_datos, text="Refuerzo:")
12450 label_agregar_refuerzo.place(x=20, y=20)
12451
12452 # Agregar el ComboBox
12453 combo_agregar_refuerzo = ttk.Combobox(labelframe_asignar_datos, values=
12454 ["Refuerzo superior", "Refuerzo inferior"])
12455 combo_agregar_refuerzo.place(x=100, y=20, width=120)
12456
12457 label_agregar_barras = tk.Label(labelframe_asignar_datos, text="#Barras:")
12458 label_agregar_barras.place(x=240, y=20)
12459
12460 # Agregar el ComboBox para barras
12461 combo_agregar_barras = ttk.Combobox(labelframe_asignar_datos, values=["#2",
12462 "#3", "#4", "#5", "#6", "#7", "#8", "#9", "#10", "#11", "#14", "#18"])
12463 combo_agregar_barras.place(x=300, y=20, width=50)
12464
12465 label_agregar_filas = tk.Label(labelframe_asignar_datos, text="#Filas:")
12466 label_agregar_filas.place(x=380, y=20)
12467
12468 combo_agregar_filas = ttk.Combobox(labelframe_asignar_datos, values=["1", "2"])
12469 combo_agregar_filas.place(x=440, y=20, width=50)
12470
12471 label_cantidad_barras = tk.Label(labelframe_asignar_datos, text="Fila 1 + Ref")
12472 label_cantidad_barras.place(x=20, y=80)
12473
12474 entry_cantidad_barras = tk.Entry(labelframe_asignar_datos)
12475 entry_cantidad_barras.place(x=90, y=80, width=50)
12476
12477 label_barras_1 = tk.Label(labelframe_asignar_datos, text="Fila 2 + Ref")
12478 label_barras_1.place(x=20, y=110)
12479
12480 entry_barras_1 = tk.Entry(labelframe_asignar_datos)
12481 entry_barras_1.place(x=90, y=110, width=50)
12482
12483 label_ba_1 = tk.Label(labelframe_asignar_datos, text="#Barras:")
12484 label_ba_1.place(x=150, y=80)
12485
12486 entry_ba_1 = tk.Entry(labelframe_asignar_datos)
12487 entry_ba_1.place(x=200, y=80, width=50)
12488
12489 label_mas_ref_1 = tk.Label(labelframe_asignar_datos, text="+")
12490 label_mas_ref_1.place(x=260, y=80)
12491
12492 label_ba_2 = tk.Label(labelframe_asignar_datos, text="#Barras:")
12493 label_ba_2.place(x=150, y=110)
12494
12495 entry_ba_2 = tk.Entry(labelframe_asignar_datos)
12496 entry_ba_2.place(x=200, y=110, width=50)
12497
12498 label_mas_ref_2 = tk.Label(labelframe_asignar_datos, text="+")
12499 label_mas_ref_2.place(x=260, y=110)
```

```
12498
12499     entry_mas_ref_1 = tk.Entry(labelframe_asignar_datos)
12500     entry_mas_ref_1.place(x=280,y=80,width=50)
12501
12502     entry_mas_ref_2 = tk.Entry(labelframe_asignar_datos)
12503     entry_mas_ref_2.place(x=280,y=110,width=50)
12504
12505     label_mas_barra_1 = tk.Label(labelframe_asignar_datos,text="#Barras:")
12506     label_mas_barra_1.place(x=340,y=80)
12507
12508     entry_mas_barra_1 = tk.Entry(labelframe_asignar_datos)
12509     entry_mas_barra_1.place(x=390,y=80,width=50)
12510
12511     label_mas_barra_2 = tk.Label(labelframe_asignar_datos,text="#Barras:")
12512     label_mas_barra_2.place(x=340,y=110)
12513
12514     entry_mas_barra_2 = tk.Entry(labelframe_asignar_datos)
12515     entry_mas_barra_2.place(x=390,y=110,width=50)
12516
12517     frame_info_barras_ref =
12518 tk.Frame(labelframe_asignar_datos,bd=2,relief="sunken")
12519     frame_info_barras_ref.place(x=20,y=140,width=460,height=50)
12520
12521     label_info_barras_ref = tk.Label(frame_info_barras_ref,text="",
12522                                     wraplength=310,
12523                                     justify="left",
12524                                     anchor="w")
12525     label_info_barras_ref.place(x=5,y=1)
12526
12527     label_info_barras_inferior = tk.Label(frame_info_barras_ref,text="",
12528                                           wraplength=310,
12529                                           justify="left",
12530                                           anchor="w")
12531     label_info_barras_inferior.place(x=5,y=20)
12532
12533     boton_aceptar_ref =
12534 ttk.Button(subventana_asignar,text="Aceptar",command=actualizar_info_barras)
12535     boton_aceptar_ref.place(x=90,y=230,width=100)
12536
12537     boton_asignar_ref =
12538 ttk.Button(subventana_asignar,text="Asignar",command=asignar_datos_treeview)
12539     boton_asignar_ref.place(x=220,y=230,width=100)
12540
12541     boton_cancelar_ref =
12542 ttk.Button(subventana_asignar,text="Cancelar",command=cerrar_subventana_armado)
12543     boton_cancelar_ref.place(x=350,y=230,width=100)
12544
12545     # Configurar el evento de cierre para restablecer la bandera
12546     subventana_asignar.protocol("WM_DELETE_WINDOW", cerrar_subventana_armado)
12547
12548     entry_ba_1.bind("<Button-1", lambda event: actualizar_entry_1(event,
12549 entry_ba_1))
12550     entry_ba_2.bind("<Button-1", lambda event: actualizar_entry_1(event,
12551 entry_ba_2))
12552     entry_mas_barra_1.bind("<Button-1", lambda event: actualizar_entry_1(event,
12553 entry_mas_barra_1))
12554     entry_mas_barra_2.bind("<Button-1", lambda event: actualizar_entry_1(event,
12555 entry_mas_barra_2))
12556
```

```
12549 # Función para actualizar la información de barras y los Entry en
12550 # labelframe_reporte
12551 def actualizar_info_barras():
12552     # Obtener el valor seleccionado en los ComboBoxes
12553     refuerzo_r = combo_agregar_refuerzo.get()
12554     filas_r = combo_agregar_filas.get()
12555
12556     # Inicializar un string vacío para almacenar el mensaje
12557     mensaje_r = ""
12558
12559     # Verificar si estamos en la fila 1
12560     if filas_r == "1":
12561         # Obtener los valores de los Entry relacionados con la fila 1
12562         cantidad_barras_r = entry_cantidad_barras.get()
12563         barras_1_r = entry_ba_1.get()
12564         mas_ref_1_r = entry_mas_ref_1.get()
12565         mas_barra_1_r = entry_mas_barra_1.get()
12566
12567         # Si solo hay datos en entry_cantidad_barras y entry_ba_1
12568         if cantidad_barras_r and barras_1_r:
12569             mensaje_r = f"{refuerzo_r}: Se escoge para la fila ({filas_r})
12570 {cantidad_barras_r}{barras_1_r}"
12571
12572         # Si también hay datos en entry_mas_ref_1 y entry_mas_barra_1
12573         if mas_ref_1_r and mas_barra_1_r:
12574             mensaje_r = f"{refuerzo_r}: Se escoge para la fila ({filas_r})
12575 {cantidad_barras_r}{barras_1_r} + {mas_ref_1_r}{mas_barra_1_r}"
12576
12577     # Actualizar los Entry en labelframe_reporte para refuerzo superior o
12578 inferior
12579     if refuerzo_r == "Refuerzo superior":
12580         entry_numero_1.delete(0, tk.END)
12581         entry_numero_1.insert(0, cantidad_barras_r)
12582
12583         entry_fila_barra_1.delete(0, tk.END)
12584         entry_fila_barra_1.insert(0, barras_1_r)
12585
12586         entry_numero_2.delete(0, tk.END)
12587         entry_numero_2.insert(0, mas_ref_1_r)
12588
12589         entry_fila_barra_2.delete(0, tk.END)
12590         entry_fila_barra_2.insert(0, mas_barra_1_r)
12591
12592     # Actualizamos el label_info_barras_ref para refuerzo superior o inferior
12593     if refuerzo_r == "Refuerzo superior":
12594         label_info_barras_ref.config(text=mensaje_r)
12595
12596     elif refuerzo_r == "Refuerzo inferior":
12597         label_info_barras_inferior.config(text=mensaje_r)
12598
12599     # **Nueva parte**: Actualizar los Entry de Refuerzo inferior también
12600 cuando es fila 1
12601     if refuerzo_r == "Refuerzo inferior":
12602         entry_fila_inferior_1.delete(0, tk.END)
12603         entry_fila_inferior_1.insert(0, barras_1_r)
12604
12605         entry_fila_inferior_2.delete(0, tk.END)
12606         entry_fila_inferior_2.insert(0, mas_barra_1_r)
12607
12608         entry_numero_inferior_1.delete(0, tk.END)
```

```

12604         entry_numero_inferor_1.insert(0, cantidad_barras_r)
12605
12606         entry_numero_inferor_2.delete(0, tk.END)
12607         entry_numero_inferor_2.insert(0, mas_ref_1_r)
12608
12609         # Verificar si estamos en la fila 2
12610         elif filas_r == "2":
12611             # Obtener los valores de los Entry relacionados con la fila 2
12612             barras_2_r = entry_barras_1.get()
12613             ba_2_r = entry_ba_2.get()
12614             mas_ref_2_r = entry_mas_ref_2.get()
12615             mas_barra_2_r = entry_mas_barra_2.get()
12616
12617             # Si solo hay datos en entry_barras_1 y entry_ba_2
12618             if barras_2_r and ba_2_r:
12619                 mensaje_r = f"{refuerzo_r}: Se escoge para la fila ({filas_r})
12620 {barras_2_r}{ba_2_r}"
12621
12622             # Si también hay datos en entry_mas_ref_2 y entry_mas_barra_2
12623             if mas_ref_2_r and mas_barra_2_r:
12624                 mensaje_r = f"{refuerzo_r}: Se escoge para la fila ({filas_r})
12625 {barras_2_r}{ba_2_r} + {mas_ref_2_r}{mas_barra_2_r}"
12626
12627             # Actualizar los Entry en labelframe_reporte para refuerzo superior o
12628             inferior
12629             if refuerzo_r == "Refuerzo superior":
12630                 entry_numero2_superior.delete(0, tk.END)
12631                 entry_numero2_superior.insert(0, barras_2_r) # Actualizamos
12632                 barras_2_r
12633
12634                 entry_fila2_superior.delete(0, tk.END)
12635                 entry_fila2_superior.insert(0, ba_2_r) # Actualizamos ba_2_r
12636
12637                 entry_numero2_combinar.delete(0, tk.END)
12638                 entry_numero2_combinar.insert(0, mas_ref_2_r) # Actualizamos
12639                 mas_ref_2_r
12640
12641                 entry_fila2_combinar.delete(0, tk.END)
12642                 entry_fila2_combinar.insert(0, mas_barra_2_r) # Actualizamos
12643                 mas_barra_2_r
12644
12645             # Actualizamos el label_info_barras_ref para refuerzo superior
12646             if refuerzo_r == "Refuerzo superior":
12647                 label_info_barras_ref.config(text=mensaje_r)
12648
12649             # Actualizamos el label_info_barras_inferior para refuerzo inferior
12650             elif refuerzo_r == "Refuerzo inferior":
12651                 label_info_barras_inferior.config(text=mensaje_r)
12652
12653             #**Nueva parte**: Actualizar los Entry de Refuerzo inferior también cuando
12654             es fila 2
12655             if refuerzo_r == "Refuerzo inferior":
12656                 entry_fila2_inferior.delete(0, tk.END)
12657                 entry_fila2_inferior.insert(0, ba_2_r)
12658
12659                 entry_numero1_inferor.delete(0, tk.END)
12660                 entry_numero1_inferor.insert(0, barras_2_r)
12661
12662                 entry_inferior2_combinar.delete(0, tk.END)
12663                 entry_inferior2_combinar.insert(0, mas_barra_2_r)

```

```

12657
12658     entry_numero1_combinar.delete(0, tk.END)
12659     entry_numero1_combinar.insert(0, mas_ref_2_r)
12660
12661
12662 def asignar_datos_treeview():
12663     # Obtener el valor seleccionado en los ComboBoxes
12664     refuerzo_r = combo_agregar_refuerzo.get() # "Refuerzo superior" o "Refuerzo
inferior"
12665     filas_r = combo_agregar_filas.get() # "1" o "2"
12666
12667     # Asegurarse de que las filas 0 y 1 existan
12668     if len(treeview_armado.get_children()) == 0: # Si no hay filas en el
treeview, insertar las filas iniciales
12669         treeview_armado.insert("", "end", values=("Superior", "", "", "", ""))
12670         treeview_armado.insert("", "end", values=("Inferior", "", "", "", ""))
12671
12672     # Verificar si la fila seleccionada es la 1
12673     if filas_r == "1":
12674         if refuerzo_r == "Refuerzo superior":
12675             # Obtener los valores de los Entry relacionados con Refuerzo superior
y Fila 1
12676             barras_1_r = entry_fila_barra_1.get()
12677             cantidad_barras_r = entry_numero_1.get()
12678             barras_2_r = entry_fila_barra_2.get()
12679             cantidad_barras_2_r = entry_numero_2.get()
12680
12681             # Buscar la fila "Superior" (fila 0)
12682             hijos_treeview = treeview_armado.get_children()
12683             if len(hijos_treeview) > 0: # Verificar que la fila "Superior" existe
12684                 fila_encontrada = False
12685                 current_values = treeview_armado.item(hijos_treeview[0])["values"]
# Acceder a la fila "Superior"
12686
12687             # Si la fila "Superior" tiene datos en la segunda columna
(barras), actualizarla
12688             if current_values[0] == "Superior":
12689                 # Si ya tiene un valor en la columna de barras, actualízalo
12690                 treeview_armado.item(hijos_treeview[0], values=("Superior", f"
{barras_1_r} {barras_2_r}", f"{cantidad_barras_r}{barras_1_r}
{cantidad_barras_2_r}{barras_2_r}", "", ""))
12691                 fila_encontrada = True
12692
12693             if not fila_encontrada:
12694                 print("Error: No se encontró la fila correspondiente para
Refuerzo superior y Fila 1.")
12695
12696             elif refuerzo_r == "Refuerzo inferior":
12697                 # Obtener los valores de los Entry relacionados con Refuerzo inferior
y Fila 1
12698                 barras_1_inferior = entry_fila_inferior_1.get()
12699                 cantidad_barras_inferior = entry_numero_inferior_1.get()
12700                 barras_2_inferior = entry_fila_inferior_2.get()
12701                 cantidad_barras_2_inferior = entry_numero_inferior_2.get()
12702
12703                 # Buscar la fila "Inferior" (fila 1)
12704                 hijos_treeview = treeview_armado.get_children()
12705                 if len(hijos_treeview) > 1: # Verificar que la fila "Inferior" existe
12706                     fila_encontrada = False

```

```
12707         current_values = treeview_armado.item(hijos_treeview[1])["values"]
12708         # Acceder a la fila "Inferior"
12709         # Si la fila "Inferior" tiene datos en la segunda columna
12710         (barras), actualizarla
12711         if current_values[0] == "Inferior":
12712             # Si ya tiene un valor en la columna de barras, actualízalo
12713             treeview_armado.item(hijos_treeview[1], values=("Inferior", f"
12714             {barras_1_inferior} {barras_2_inferior}", f"{cantidad_barras_inferior}
12715             {barras_1_inferior} {cantidad_barras_2_inferior}{barras_2_inferior}", "", ""))
12716             fila_encontrada = True
12717
12718         if not fila_encontrada:
12719             print("Error: No se encontró la fila correspondiente para
12720             Refuerzo inferior y Fila 1.")
12721
12722         # Actualizar el mensaje en el label después de actualizar
12723         label_asignar_check.config(text="¡Muy bien! Has completado esta fase
12724         con éxito. Ahora, por favor, continúa con la configuración del refuerzo
12725         transversal para asegurar que todo esté correctamente ajustado")
12726
12727 # Función para actualizar el Entry cuando se hace clic
12728 def actualizar_entry_1(event, entry_asociacion):
12729     # Obtener el valor seleccionado en el ComboBox
12730     valor_seleccionado_1 = combo_agregar_barras.get()
12731
12732     # Borrar cualquier texto existente y establecer el valor en el Entry
12733     correspondiente
12734     entry_asociacion.delete(0, tk.END)
12735     entry_asociacion.insert(0, valor_seleccionado_1)
12736
12737 def Dimensionamiento_estribos(event, entry_prueba_seccion_estribo,
12738 combo_seccion_estribo):
12739     # Obtener el valor del Entry
12740     dato_entry = entry_prueba_seccion_estribo.get()
12741
12742     # Actualizar el Combobox con el valor del Entry
12743     combo_seccion_estribo['values'] = (dato_entry,)
12744
12745     # Establecer el valor en el Combobox
12746     combo_seccion_estribo.set(dato_entry)
12747
12748 def Elementos_confinados(event, combo_seccion_barra, Entry_doble,
12749 entry_prueba_seccion_estribo, entry_prueba_seccion_recubrimiento,
12750 entry_A_confi, entry_C_confi, entry_B_confi,
12751 entry_D_confi,entry_prueba_estribo_diseño):
12752     # Obtener la selección del Combobox
12753     seleccion = combo_seccion_barra.get()
12754
12755     # Definir los diámetros según la selección
12756     if seleccion == "#3":
12757         diametro_barra_3 = 9.5
12758         doblez_diametro = 4 * diametro_barra_3
12759     elif seleccion == "#4":
12760         diametro_barra_4 = 12.7
12761         doblez_diametro = 4 * diametro_barra_4
12762     else:
12763         return
12764
12765     # Calcular el resultado y convertirlo a centímetros
```

```

12756 resultado_doblez = doblez_diametro / 10
12757
12758 # Actualizar el Entry con el resultado formateado
12759 Entry_doblez.delete(0, tk.END)
12760 Entry_doblez.insert(0, f'{resultado_doblez} cm')
12761
12762 # Actualizar el Entry de "prueba estribo diseño" con la selección del Combobox
12763 entry_prueba_estribo_diseño.delete(0, tk.END)
12764 entry_prueba_estribo_diseño.insert(0, seleccion)
12765
12766 # Obtener los valores del entry_prueba_seccion_estribo (Ejemplo: "VG 40.0 X
12767 50.0")
12768 texto_estribo_1 = entry_prueba_seccion_estribo.get()
12769
12770 # Filtrar los números de la cadena de texto y extraer A_estribo y C_estribo
12771 (ahora incluye flotantes)
12772 numeros_1 = [float(s) for s in texto_estribo_1.split() if s.replace('.', '',
12773 1).isdigit()]
12774
12775 # Verificar si se encontraron al menos dos números
12776 if len(numeros_1) >= 2:
12777     A_estribo = numeros_1[0] # Primer número
12778     C_estribo = numeros_1[1] # Segundo número
12779 else:
12780     # Si no se encuentran dos números, actualizar los Entry como vacío
12781     entry_A_confi.delete(0, tk.END)
12782     entry_C_confi.delete(0, tk.END)
12783     return
12784
12785 # Obtener el valor de recubrimiento (en cm)
12786 try:
12787     Recu_estribo = float(entry_prueba_seccion_recubrimiento.get())
12788 except ValueError:
12789     # Si el valor de recubrimiento no es válido, no hacer nada
12790     return
12791
12792 # Calcular las dimensiones del estribo
12793 ancho_dimension_estribo = A_estribo - 2 * Recu_estribo * 100 # Convertir
12794 recubrimiento de metros a centímetros
12795 altura_dimension_estribo = C_estribo - 2 * Recu_estribo * 100
12796
12797 # Actualizar los resultados en los Entry correspondientes con "cm"
12798 entry_A_confi.delete(0, tk.END)
12799 entry_A_confi.insert(0, f'{ancho_dimension_estribo} cm')
12800
12801 entry_C_confi.delete(0, tk.END)
12802 entry_C_confi.insert(0, f'{altura_dimension_estribo} cm')
12803
12804 # Actualizar los nuevos Entry (entry_B_confi y entry_D_confi)
12805 entry_B_confi.delete(0, tk.END)
12806 entry_B_confi.insert(0, f'{ancho_dimension_estribo} cm')
12807
12808 entry_D_confi.delete(0, tk.END)
12809 entry_D_confi.insert(0, f'{altura_dimension_estribo} cm')
12810
12811 def Extension_corte_estribo(event, combo_gancho_estribo, combo_seccion_barra,
12812 entry_G_confi, label_ok):
12813     # Verificar si el valor seleccionado en el combo_gancho_estribo es "135°"
12814     if combo_gancho_estribo.get() == "135°":
12815         # Obtener la selección del combo_seccion_barra

```

```
12811     seleccion = combo_seccion_barra.get()
12812
12813     # Definir los diámetros según la selección del combo_seccion_barra
12814     if seleccion == "#3":
12815         corte_diametro_3 = 9.5
12816         corte_estribo = 6 * corte_diametro_3
12817     elif seleccion == "#4":
12818         corte_diametro_4 = 12.7
12819         corte_estribo = 6 * corte_diametro_4
12820     else:
12821         # Si no hay una selección válida, no hacer nada
12822         return
12823
12824     # Verificar si el resultado es menor que 75, en cuyo caso se asigna 75
12825     if corte_estribo < 75:
12826         corte_estribo = 75
12827     # Dividir el resultado entre 10
12828     corte_estribo = corte_estribo / 10
12829
12830     # Mostrar el resultado en el entry_G_confi correspondiente
12831     entry_G_confi.delete(0, tk.END)
12832     entry_G_confi.insert(0, f"{corte_estribo:.2f} cm")
12833     # Actualizar el label_ok con "Ok!"
12834     label_ok.config(text="OK!", fg="green")
12835
12836 def cerrar_subventana_armado():
12837     global subventana_asignar_abierta
12838     subventana_asignar.destroy()
12839     subventana_asignar_abierta = False
12840
12841
12842
12843 frame_ventana_flexion= tk.Frame(ventana,bd=10,relief="ridge",bg="white")
12844 frame_ventana_flexion.place(x=838,y=35,width=660,height=737)
12845
12846 frame_interno = tk.Frame(frame_ventana_flexion, bd=2, relief="flat")
12847 frame_interno.pack(fill="both", expand=True)
12848
12849
12850 # Crear el estilo para personalizar las pestañas
12851 style = ttk.Style()
12852
12853 # Estilo para el Notebook
12854 style.configure("CustomNotebook.TNotebook",
12855                 background="lightgray",
12856                 tabmargins=[0, 0, 0, 0])
12857
12858 # Estilo para las pestañas
12859 style.configure("CustomNotebook.TNotebook.Tab",
12860                 background="white",
12861                 foreground="black",
12862                 padding=[10, 2],
12863                 font=("Arial", 9))
12864
12865 # Estilo para la pestaña seleccionada
12866 style.map("CustomNotebook.TNotebook.Tab",
12867           background=[("selected", "lightgray")],
12868           foreground=[("selected", "black")])
12869
12870 # Crear el Notebook dentro de frame_interno con el estilo personalizado
```

```
12871 notebook_flexion = ttk.Notebook(frame_interno, style="CustomNotebook.TNotebook")
12872
12873
12874 # Crear las pestañas del Notebook
12875 tab1_flexion = tk.Frame(notebook_flexion)
12876 tab2_cortante = tk.Frame(notebook_flexion)
12877
12878 # Agregar las pestañas al Notebook
12879 notebook_flexion.add(tab1_flexion, text="Diseño flexión")
12880 notebook_flexion.add(tab2_cortante, text="Diseño cortante")
12881
12882 frame_ventana_flexion = tk.Frame(tab1_flexion,bd=2,relief="raise")
12883 frame_ventana_flexion.place(x=20,y=20,width=590,height=500)
12884
12885 frame_interno_1 = tk.Frame(tab1_flexion, bd=2, relief="raise")
12886 frame_interno_1.place(x=20,y=530,width=590,height=150)
12887
12888 # Crear el Notebook dentro del Frame
12889 notebook_reporte = ttk.Notebook(frame_interno_1,style="CustomNotebook.TNotebook")
12890 notebook_reporte.pack(fill="both", expand=True)
12891
12892 # Crear la pestaña
12893 tab_reporte = tk.Frame(notebook_reporte) # Pestaña vacía
12894 notebook_reporte.add(tab_reporte, text="Reporte de resultados")
12895
12896 labelframe_reporte = tk.LabelFrame(tab_reporte,text="Resultados de diseño
flexión")
12897 labelframe_reporte.place(x=10,y=10,width=560,height=100)
12898
12899 frame_reporte = tk.Frame(labelframe_reporte,bd=2,relief="flat")
12900 frame_reporte.place(x=10,y=10,width=30,height=30)
12901
12902 frame_reporte_pdf = tk.Frame(labelframe_reporte,bd=2,relief="flat")
12903 frame_reporte_pdf.place(x=10,y=45,width=30,height=30)
12904
12905 # Cargar la imagen con el nombre 'image_report'
12906 image_report = Image.open("Iconos_proyecto/Doc_word.png")
12907 image_report_pdf = Image.open("Iconos_proyecto/Doc_pdf.png")
12908 # Redimensionar la imagen si es necesario (ajustarlo al tamaño del frame)
12909 image_doc = image_report.resize((30, 30))
12910 image_pdf = image_report_pdf.resize((30, 30))
12911 # Convertir la imagen a un formato que Tkinter pueda usar
12912 photo_report = ImageTk.PhotoImage(image_doc)
12913 photo_report_pdf = ImageTk.PhotoImage(image_pdf)
12914 # Insertar la imagen en el frame usando un Label
12915 label_report = tk.Label(frame_reporte, image=photo_report)
12916 label_report.pack()
12917 label_report_pdf = tk.Label(frame_reporte_pdf, image= photo_report_pdf)
12918 label_report_pdf.pack()
12919 # Mantener una referencia a la imagen
12920 label_report.image = photo_report
12921 label_report_pdf.image = photo_report_pdf
12922
12923 # Crear las variables para los checkbuttons
12924 check_reporte_word = tk.BooleanVar()
12925 check_reporte_pdf = tk.BooleanVar()
12926 # Crear los checkbuttons
12927 checkbutton_word = tk.Checkbutton(labelframe_reporte, text="Generar reporte de
resultados - formato Word", variable=check_reporte_word,command=Generar_reportes)
12928 checkbutton_word.place(x=50,y=10)
```

```
12929 checkbutton_pdf = tk.Checkbutton(labelframe_reporte, text="Generar reporte de
resultados - formato Pdf", variable=check_reporte_pdf)
12930 checkbutton_pdf.place(x=50,y=45)
12931 checkbutton_pdf.config(state="disabled")
12932
12933 '*****Resultados diseño
cortante*****'
12934
12935 frame_ventana_cortante = tk.Frame(tab2_cortante,bd=2,relief="raise")
12936 frame_ventana_cortante.place(x=20,y=20,width=590,height=500)
12937
12938 frame_interno_2 = tk.Frame(tab2_cortante, bd=2, relief="raise")
12939 frame_interno_2.place(x=20,y=530,width=590,height=150)
12940
12941 # Crear el Notebook dentro del Frame
12942 notebook_reporte_1 =
ttk.Notebook(frame_interno_2,style="CustomNotebook.TNotebook")
12943 notebook_reporte_1.pack(fill="both", expand=True)
12944
12945 # Crear la pestaña
12946 tab_reporte_1 = tk.Frame(notebook_reporte_1) # Pestaña vacía
12947 notebook_reporte_1.add(tab_reporte_1, text="Reporte de resultados")
12948
12949 labelframe_reporte_1 = tk.LabelFrame(tab_reporte_1,text="Resultados de diseño
cortante")
12950 labelframe_reporte_1.place(x=10,y=10,width=560,height=100)
12951
12952 frame_reporte_1 = tk.Frame(labelframe_reporte_1,bd=2,relief="flat")
12953 frame_reporte_1.place(x=10,y=10,width=30,height=30)
12954
12955 frame_reporte_pdf_1 = tk.Frame(labelframe_reporte_1,bd=2,relief="flat")
12956 frame_reporte_pdf_1.place(x=10,y=45,width=30,height=30)
12957
12958 # Cargar la imagen con el nombre 'image_report'
12959 image_report_1 = Image.open("Iconos_proyecto/Doc_word.png")
12960 image_report_pdf_1 = Image.open("Iconos_proyecto/Doc_pdf.png")
12961 # Redimensionar la imagen si es necesario (ajustarlo al tamaño del frame)
12962 image_doc_1 = image_report_1.resize((30, 30))
12963 image_pdf_1 = image_report_pdf_1.resize((30, 30))
12964 # Convertir la imagen a un formato que Tkinter pueda usar
12965 photo_report_1 = ImageTk.PhotoImage(image_doc_1)
12966 photo_report_pdf_1 = ImageTk.PhotoImage(image_pdf_1)
12967 # Insertar la imagen en el frame usando un Label
12968 label_report_1 = tk.Label(frame_reporte_1, image=photo_report_1)
12969 label_report_1.pack()
12970 label_report_pdf_1 = tk.Label(frame_reporte_pdf_1, image= photo_report_pdf_1)
12971 label_report_pdf_1.pack()
12972 # Mantener una referencia a la imagen
12973 label_report_1.image = photo_report_1
12974 label_report_pdf_1.image = photo_report_pdf_1
12975
12976 # Crear las variables para los checkbuttons
12977 check_reporte_word_1 = tk.BooleanVar()
12978 check_reporte_pdf_1 = tk.BooleanVar()
12979 # Crear los checkbuttons
12980 checkbutton_word_1 = tk.Checkbutton(labelframe_reporte_1, text="Generar reporte de
resultados - formato Word",
variable=check_reporte_word_1,command=generar_reporte_cortante)
12981 checkbutton_word_1.place(x=50,y=10)
```

```
12982 checkbutton_pdf_1 = tk.Checkbutton(labelframe_reporte_1, text="Generar reporte de
resultados - formato Pdf", variable=check_reporte_pdf_1)
12983 checkbutton_pdf_1.place(x=50,y=45)
12984 checkbutton_pdf_1.config(state="disabled")
12985
12986 #entrys de prueba para refuerzo superior fila 1
12987 entry_fila_barra_1 = tk.Entry(labelframe_reporte)
12988 #entry_fila_barra_1.place(x=1,y=1,width=50)
12989 entry_fila_barra_2 = tk.Entry(labelframe_reporte) # combinar numero de barras de
refuerzo
12990 #entry_fila_barra_2.place(x=100,y=1,width=50)
12991 # entrys de cantidad para cantidad superior fila 1
12992 entry_numero_1 = tk.Entry(labelframe_reporte)
12993 #entry_numero_1.place(x=1,y=20,width=50)
12994 entry_numero_2 = tk.Entry(labelframe_reporte) # combinar cantidad de barras de
refuerzo
12995 #entry_numero_2.place(x=100,y=20,width=50)
12996
12997 #entrys de prueba para refuerzo inferior fila 1
12998 entry_fila_inferior_1 = tk.Entry(labelframe_reporte)
12999 #entry_fila_inferior_1.place(x=200,y=1,width=100)
13000 entry_fila_inferior_2= tk.Entry(labelframe_reporte) # combinar numero de barras de
refuerzo
13001 #entry_fila_inferior_2.place(x=300,y=1,width=100)
13002 # entrys de cantidad para cantidad superior fila 1
13003 entry_numero_inferor_1 = tk.Entry(labelframe_reporte)
13004 #entry_numero_inferor_1.place(x=200,y=20,width=100)
13005 entry_numero_inferor_2 = tk.Entry(labelframe_reporte) # combinar cantidad de
barras de refuerzo
13006 #entry_numero_inferor_2.place(x=300,y=20,width=100)
13007
13008 #entrys de prueba para refuerzo superior fila 2
13009 entry_fila2_superior = tk.Entry(labelframe_reporte)
13010 #entry_fila2_superior.place(x=1,y=1,width=50)
13011 entry_fila2_combinar = tk.Entry(labelframe_reporte) # combinar numero de barras de
refuerzo
13012 #entry_fila2_combinar.place(x=100,y=1,width=50)
13013 # entrys de cantidad para cantidad superior fila 2
13014 entry_numero2_superior = tk.Entry(labelframe_reporte)
13015 #entry_numero2_superior.place(x=1,y=20,width=50)
13016 entry_numero2_combinar = tk.Entry(labelframe_reporte) # combinar cantidad de
barras de refuerzo
13017 #entry_numero2_combinar.place(x=100,y=20,width=50)
13018
13019 #entrys de prueba para refuerzo inferior fila 2
13020 entry_fila2_inferior = tk.Entry(labelframe_reporte)
13021 #entry_fila2_inferior.place(x=200,y=1,width=100)
13022 entry_inferior2_combinar= tk.Entry(labelframe_reporte) # combinar numero de barras
de refuerzo
13023 #entry_inferior2_combinar.place(x=300,y=1,width=100)
13024 # entrys de cantidad para cantidad superior fila 1
13025 entry_numero1_inferor = tk.Entry(labelframe_reporte)
13026 #entry_numero1_inferor.place(x=200,y=20,width=100)
13027 entry_numero1_combinar = tk.Entry(labelframe_reporte) # combinar cantidad de
barras de refuerzo
13028 #entry_numero1_combinar.place(x=300,y=20,width=100)
13029
13030 entry_prueba_seccion_estribo = tk.Entry(labelframe_reporte)
13031 #entry_prueba_seccion_estribo.place(x=300,y=20,width=100)
13032 entry_prueba_seccion_recubrimiento = tk.Entry(labelframe_reporte)
```

```
13033 #entry_prueba_seccion_recubrimiento.place(x=300,y=20,width=100)
13034 entry_prueba_seccion_base = tk.Entry(labelframe_reporte)
13035 #entry_prueba_seccion_base.place(x=300,y=20,width=100)
13036 entry_prueba_seccion_altura = tk.Entry(labelframe_reporte)
13037 #entry_prueba_seccion_altura.place(x=300,y=20,width=100)
13038 entry_prueba_estribo_diseño = tk.Entry(labelframe_reporte)
13039 #entry_prueba_estribo_diseño.place(x=300,y=20,width=100)
13040 entry_prueba_longitud = tk.Entry(labelframe_reporte)
13041 #entry_prueba_longitud.place(x=300,y=20,width=100)
13042
13043 #TRAMO 1
13044 entry_tramo_1 = tk.Entry(labelframe_reporte)
13045 #entry_tramo_1.place(x=100,y=1,width=100)
13046 entry_tramo_1_S =tk.Entry(labelframe_reporte)
13047 #entry_tramo_1_S.place(x=100,y=40,width=100)
13048
13049 #TRAMO 2
13050 entry_tramo_2 = tk.Entry(labelframe_reporte)
13051 #entry_tramo_2.place(x=200,y=1,width=100)
13052 entry_tramo_2_S =tk.Entry(labelframe_reporte)
13053 #entry_tramo_2_S.place(x=200,y=40,width=100)
13054
13055 #TRAMO 3
13056 entry_tramo_3 = tk.Entry(labelframe_reporte)
13057 #entry_tramo_3.place(x=300,y=1,width=100)
13058 entry_tramo_3_S =tk.Entry(labelframe_reporte)
13059 #entry_tramo_3_S.place(x=300,y=40,width=100)
13060
13061 #BARRAS SUPERIOR
13062 entry_barras_superior = tk.Entry(labelframe_reporte)
13063 #entry_barras_superior.place(x=100,y=1,width=100)
13064
13065 #BARRAS INFERIOR
13066 entry_barras_inferior = tk.Entry(labelframe_reporte)
13067 #entry_barras_inferior.place(x=100,y=40,width=100)
13068
13069 # Crear labels de pruebas
13070 # label para el valor profundidad "d"
13071 label_profundidad_prueba = tk.Label(frame_interno_1,bg="red")
13072 #label_profundidad_prueba.place(x=1,y=1,width=100)
13073
13074 # label para el valor de "b"
13075 label_b_prueba = tk.Label(frame_interno_1,bg="red")
13076 #label_b_prueba.place(x=100,y=1,width=100)
13077
13078 # Label para el valor de "Vmax"
13079 label_Vmax_prueba = tk.Label(frame_interno_1,bg="red")
13080 #label_Vmax_prueba.place(x=200,y=1,width=100)
13081
13082 # Crear un Canvas y agregarlo al LabelFrame
13083 canvas_flexion = tk.Canvas(frame_ventana_flexion)
13084 canvas_flexion.pack(fill="both", expand=True)
13085
13086 # label para el dato de nombre de estudiante
13087 label_estudiante = tk.Label(labelframe_reporte,text="",bg="red")
13088 #label_estudiante.place(x=10,y=1,width=100)
13089
13090 # label para el dato de nombre de modelo
13091 label_modelo = tk.Label(labelframe_reporte,text="",bg="green")
13092 #label_modelo.place(x=10,y=20,width=100)
```

```
13093
13094 # label para el dato de modulo de elasticidad del concreto
13095 label_elasticidad = tk.Label(labelframe_reporte,text="",bg="blue")
13096 #label_elasticidad.place(x=10,y=40,width=100)
13097
13098 # label para el dato de resistencia a compresion del concreto
13099 label_fc = tk.Label(labelframe_reporte,text="",bg="grey")
13100 #label_fc.place(x=10,y=1,width=100)
13101
13102 # Label para el dato de altura de la viga
13103 label_h= tk.Label(labelframe_reporte,text="",bg="orange")
13104 #label_h.place(x=10,y=40,width=100)
13105
13106 # Label para el dato del recubrimiento de la viga
13107 label_recubrimiento = tk.Label(labelframe_reporte,text="",bg="yellow")
13108 #label_recubrimiento.place(x=10,y=10,width=100)
13109
13110 # Label para el dato de la altura efectiva
13111 label_altura_efectiva = tk.Label(labelframe_reporte,text="",bg="white")
13112 #label_altura_efectiva.place(x=10,y=40,width=100)
13113
13114 # Label para el dato de la resistencia concreto
13115 label_resistencia_psi = tk.Label(labelframe_reporte,text="",bg="green")
13116 #label_resistencia_psi.place(x=10,y=20,width=100)
13117
13118 # Label para el dato de Wpp
13119 label_cargas_Wpp = tk.Label(labelframe_reporte,text="",bg="red")
13120 #label_cargas_Wpp.place(x=10,y=1,width=100)
13121
13122 # Label para el dato de WD
13123 label_cargas_WD = tk.Label(labelframe_reporte,text="",bg="blue")
13124 #label_cargas_WD.place(x=120,y=1,width=100)
13125
13126 # Label para el dato de WL
13127 label_cargas_WL = tk.Label(labelframe_reporte,text="",bg="yellow")
13128 #label_cargas_WL.place(x=230,y=1,width=100)
13129
13130 # Label para el dato de Wtotal
13131 label_cargas_Wtotal = tk.Label(labelframe_reporte,text="",bg="orange")
13132 #label_cargas_Wtotal.place(x=340,y=1,width=100)
13133
13134 # Label para la reaccion apoyo A y B
13135 label_reaccion = tk.Label(labelframe_reporte,text="",bg="red")
13136 #label_reaccion.place(x=10,y=20,width=100)
13137
13138 # Label para el dato Mmax[-]
13139 label_Mmax_1 = tk.Label(labelframe_reporte,text="",bg="red")
13140 #label_Mmax_1.place(x=10,y=1,width=100)
13141
13142 # Label para el dato Mmax[+]
13143 label_Mmax_2 = tk.Label(labelframe_reporte,text="",bg="blue")
13144 #label_Mmax_2.place(x=110,y=1,width=100)
13145
13146 # Label para el dato cuantia minima
13147 label_cuantia_minima = tk.Label(labelframe_reporte,text="",bg="green")
13148 #label_cuantia_minima.place(x=10,y=1,width=100)
13149
13150 # Label para el dato cuantia maxima
13151 label_cuantia_maxima = tk.Label(labelframe_reporte,text="",bg="grey")
13152 #label_cuantia_maxima.place(x=10,y=20,width=100)
```

```
13153
13154 # Label para el dato beta1
13155 label_beta1 = tk.Label(labelframe_reporte,text="",bg="red")
13156 #label_beta1.place(x=10,y=40,width=100)
13157
13158 # Label para el dato Rn
13159 label_Resistencia_Rn = tk.Label(labelframe_reporte,text="",bg="blue")
13160 #label_Resistencia_Rn.place(x=10,y=1,width=100)
13161
13162 # label para el dato cuantia requerida
13163 label_cuantia_requerida = tk.Label(labelframe_reporte,text="",bg="green")
13164 #label_cuantia_requerida.place(x=10,y=20,width=100)
13165
13166 # label para el dato acero requerido
13167 label_acero_requerido = tk.Label(labelframe_reporte,text="",bg="yellow")
13168 #label_acero_requerido.place(x=10,y=40,width=100)
13169
13170 # label para el dato de Rn2
13171 label_Rn2 = tk.Label(labelframe_reporte,text="",bg="white")
13172 #label_Rn2.place(x=10,y=1,width=100)
13173
13174 # label para el dato req C2
13175 label_req_C2 = tk.Label(labelframe_reporte,text="",bg="red")
13176 #label_req_C2.place(x=10,y=20,width=100)
13177
13178 # label para el dato As2
13179 label_As2 = tk.Label(labelframe_reporte,text="",bg="blue")
13180 #label_As2.place(x=10,y=40,width=100)
13181
13182 # Label para el dato estribo
13183 label_estribo = tk.Label(labelframe_reporte,text="",bg="green")
13184 #label_estribo.place(x=10,y=10,width=100)
13185
13186 #label para el dato de area estribo
13187 label_area_estribo = tk.Label(labelframe_reporte,text="",bg="orange")
13188 #label_area_estribo.place(x=10,y=50,width=100)
13189
13190 #label para el dato de  $\phi M_n1$ 
13191 label_ $\phi M_n1$  = tk.Label(labelframe_reporte,text="",bg="orange")
13192 #label_ $\phi M_n1$ .place(x=10,y=50,width=100)
13193
13194 #label para el dato de  $\phi M_n2$ 
13195 label_ $\phi M_n2$  = tk.Label(labelframe_reporte,text="",bg="blue")
13196 #label_ $\phi M_n2$ .place(x=10,y=10,width=100)
13197
13198 #label para el dato si cumple o no cumple
13199 label_conclusion_ $\phi M_n$  = tk.Label(labelframe_reporte,text="",bg="green")
13200 #label_conclusion_ $\phi M_n$ .place(x=100,y=1,width=100)
13201
13202 #label para el dato si cumple o no cumple
13203 label_conclusion_ $\phi M_n2$  = tk.Label(labelframe_reporte,text="",bg="red")
13204 #label_conclusion_ $\phi M_n2$ .place(x=100,y=20,width=100)
13205
13206 # Label para el dato No.barra
13207 label_barra = tk.Label(labelframe_reporte,text="",bg="yellow")
13208 #label_barra.place(x=1,y=1,width=100)
13209
13210 # Label para el dato info_barra
13211 label_info_b = tk.Label(labelframe_reporte,text="",bg="red")
13212 #label_info_b.place(x=1,y=20,width=100)
```

```
13213
13214 # label para el dato area barra
13215 label_area_barra = tk.Label(labelframe_reporte,text="",bg="blue")
13216 #label_area_barra.place(x=1,y=50,width=100)
13217
13218 # label para el dato diametro barra
13219 label_Øbarras = tk.Label(labelframe_reporte,text="",bg="green")
13220 #label_Øbarras.place(x=110,y=50,width=100)
13221
13222 # label para el dato barra flexion
13223 label_barra_flexion = tk.Label(labelframe_reporte,text="",bg="orange")
13224 #label_barra_flexion.place(x=1,y=10,width=100)
13225
13226 # label para el dato de barra flexion1
13227 label_barra_flexion1 = tk.Label(labelframe_reporte,text="",bg="yellow")
13228 #label_barra_flexion1.place(x=1,y=50,width=100)
13229
13230 # Label parametro_A_1
13231 label_A_1 = tk.Label(labelframe_reporte,text="",bg="red")
13232 #label_A_1.place(x=1,y=1,width=100)
13233
13234 # Label parametro_C_1
13235 label_C_1 = tk.Label(labelframe_reporte,text="",bg="green")
13236 #label_C_1.place(x=1,y=50,width=100)
13237
13238 # label parametro_D_1
13239 label_D_1 = tk.Label(labelframe_reporte,text="",bg="orange")
13240 #label_D_1.place(x=100,y=1,width=100)
13241
13242 # label parametro_n_1
13243 label_n_1 = tk.Label(labelframe_reporte,text="",bg="yellow")
13244 #label_n_1.place(x=100,y=20,width=100)
13245
13246 # label paremetro dp_1
13247 label_dp_1 = tk.Label(labelframe_reporte,text="",bg="white")
13248 #label_dp_1.place(x=100,y=50,width=100)
13249
13250 # label base requerida
13251 label_base_req = tk.Label(labelframe_reporte,text="",bg="red")
13252 #label_base_req.place(x=1,y=1,width=100)
13253
13254 # label base requerida1
13255 label_base_req1 = tk.Label(labelframe_reporte,text="",bg="blue")
13256 #label_base_req1.place(x=1,y=20,width=100)
13257
13258 # label Mcr
13259 label_Mcr_1 = tk.Label(labelframe_reporte,text="",bg="green")
13260 #label_Mcr_1.place(x=1,y=1,width=100)
13261
13262 # label Mcr_check
13263 label_Mcr_check = tk.Label(labelframe_reporte,text="",bg="yellow")
13264 #label_Mcr_check.place(x=1,y=20,width=100)
13265
13266 # label Rn_mcr
13267 label_Rn_mcr = tk.Label(labelframe_reporte,text="",bg="orange")
13268 #label_Rn_mcr.place(x=1,y=50,width=100)
13269
13270 # label Rn_mcr_check
13271 label_Rn_mcr_check = tk.Label(labelframe_reporte,text="",bg="white")
13272 #label_Rn_mcr_check.place(x=120,y=50,width=100)
```

```
13273
13274 # label preq_mcr
13275 label_preq_mcr = tk.Label(labelframe_reporte,text="",bg="red")
13276 #label_preq_mcr.place(x=200,y=50,width=100)
13277
13278 # label info_barras_mcr
13279 label_info_barras_mcr = tk.Label(labelframe_reporte,text="",bg="blue")
13280 #label_info_barras_mcr.place(x=1,y=1,width=100)
13281
13282 # label area_req_mcr
13283 label_area_req_mcr = tk.Label(labelframe_reporte,text="",bg="yellow")
13284 #label_area_req_mcr.place(x=1,y=30,width=100)
13285
13286 # label area_barra_mcr
13287 label_area_barra_mcr = tk.Label(labelframe_reporte,text="",bg="green")
13288 #label_area_barra_mcr.place(x=1,y=50,width=100)
13289
13290 # label num_barra_mcr
13291 label_num_barra_mcr = tk.Label(labelframe_reporte,text="",bg="orange")
13292 #label_num_barra_mcr.place(x=110,y=50,width=100)
13293
13294 # label diametro_traccion
13295 label_diametro_traccion = tk.Label(labelframe_reporte,text="",bg="red")
13296 #label_diametro_traccion.place(x=1,y=1,width=100)
13297
13298 # label Ld_traccion
13299 label_Ld_traccion = tk.Label(labelframe_reporte,text="",bg="blue")
13300 #label_Ld_traccion.place(x=1,y=30,width=100)
13301
13302 # label diametro_compresion
13303 label_diametro_compresion = tk.Label(labelframe_reporte,text="",bg="green")
13304 #label_diametro_compresion.place(x=1,y=55,width=100)
13305
13306 # label diametro Ldc
13307 label_Ldc_compresion = tk.Label(labelframe_reporte,text="",bg="yellow")
13308 #label_Ldc_compresion.place(x=110,y=55,width=100)
13309
13310 # label ldh_t
13311 label_ldh_t = tk.Label(labelframe_reporte,text="",bg="white")
13312 #label_ldh_t.place(x=220,y=55,width=100)
13313
13314 # label ldh_c
13315 label_ldh_c = tk.Label(labelframe_reporte,text="",bg="grey")
13316 #label_ldh_c.place(x=330,y=55,width=100)
13317
13318 # label gancho
13319 label_gancho_anclaje = tk.Label(labelframe_reporte,text="",bg="pink")
13320 #label_gancho_anclaje.place(x=440,y=55,width=100)
13321
13322 # label longitud_t
13323 label_longitud_t = tk.Label(labelframe_reporte,text="",bg="red")
13324 #label_longitud_t.place(x=1,y=1,width=100)
13325
13326 # label longitud_c
13327 label_longitud_c = tk.Label(labelframe_reporte,text="",bg="blue")
13328 #label_longitud_c.place(x=1,y=20,width=100)
13329
13330 # Label ancho_columna
13331 label_ancho_columna = tk.Label(labelframe_reporte,text="",bg="green")
13332 #label_ancho_columna.place(x=1,y=50,width=100)
```

```
13333
13334 # numero de ramas
13335 label_num_ramas = tk.Label(labelframe_reporte,text="",bg="red")
13336 #label_num_ramas.place(x=1,y=1,width=100)
13337
13338 # Smax_estribos
13339 Smax_estribos = tk.Label(labelframe_reporte,text="",bg="blue")
13340 #Smax_estribos.place(x=100,y=1,width=100)
13341
13342 # Av_area requerida
13343 label_Av_requerida = tk.Label(labelframe_reporte,text="",bg="red")
13344 #label_Av_requerida.place(x=210,y=1,width=100)
13345
13346 # Vs_resistencia_req
13347 Vs_resistencia_req = tk.Label(labelframe_reporte,text="",bg="green")
13348 #Vs_resistencia_req.place(x=320,y=1,width=100)
13349
13350 # label_Vu
13351 label_Vu = tk.Label(labelframe_reporte,text="",bg="blue")
13352 #label_Vu.place(x=1,y=20,width=100)
13353
13354 # label_φVc
13355 label_φVc = tk.Label(labelframe_reporte,text="",bg="green")
13356 #label_φVc.place(x=110,y=20,width=100)
13357
13358 # label_φVc/2
13359 label_φVc_2 = tk.Label(labelframe_reporte,text="",bg="yellow")
13360 #label_φVc_2.place(x=220,y=20,width=100)
13361
13362 # entry_base_triangulo
13363 entry_base_triangulo = tk.Entry(labelframe_reporte)
13364 #entry_base_triangulo.place(x=100,y=20,width=100)
13365
13366 #entry_S
13367 entry_S = tk.Entry(labelframe_reporte)
13368 #entry_S.place(x=10,y=10)
13369
13370 # label Vs_resistencia
13371 Vs_resistencia = tk.Label(labelframe_reporte,text="",bg="yellow")
13372 #Vs_resistencia.place(x=220,y=20,width=100)
13373
13374 # labe_check_Vs
13375 label_check_Vs = tk.Label(labelframe_reporte,text="",bg="red")
13376 #label_check_Vs.place(x=220,y=20,width=100)
13377
13378 # label_check_control
13379 label_check_control = tk.Label(labelframe_reporte,text="",bg="blue")
13380 #label_check_control.place(x=100,y=20,width=100)
13381
13382 # Label_check_ramas
13383 Label_check_ramas = tk.Label(labelframe_reporte,text="",bg="blue")
13384 #Label_check_ramas.place(x=100,y=20,width=100)
13385
13386 # Label_check_separacion
13387 Label_check_separacion = tk.Label(labelframe_reporte,text="",bg="red")
13388 #Label_check_separacion.place(x=220,y=20,width=100)
13389
13390 #label_φVn
13391 label_φVn = tk.Label(labelframe_reporte,text="",bg="red")
13392 #label_φVn.place(x=100,y=20,width=100)
```

```

13393
13394 #label_Vu_φVn
13395 label_Vu_φVn = tk.Label(labelframe_reporte,text="",bg="green")
13396 #label_Vu_φVn.place(x=210,y=20,width=100)
13397
13398 #label_check_Vu_φVn
13399 label_check_Vu_φVn= tk.Label(labelframe_reporte,text="",bg="blue")
13400 #label_check_Vu_φVn.place(x=100,y=20,width=100)
13401
13402 # Crear un Canvas y agregarlo al LabelFrame
13403 canvas_cortante_1 = tk.Canvas(frame_ventana_cortante)
13404 canvas_cortante_1.pack(fill="both", expand=True)
13405
13406 # Crear un Scrollbar vertical
13407 scrollbar_y_1 = tk.Scrollbar(frame_ventana_cortante, orient="vertical",
13408                             command=canvas_cortante_1.yview)
13409 scrollbar_y_1.place(x=565,y=1,width=20,height=495)
13410
13410 # Vincular el Scrollbar vertical al Canvas
13411 canvas_cortante_1.config(yscrollcommand=scrollbar_y_1.set)
13412
13413 # Añadir contenido al Canvas_cortante
13414 labelframe_cortante_verificar = tk.LabelFrame(canvas_cortante_1,text="Análisis de
13415 resistencia a cortante [V]")
13415 labelframe_cortante_check = tk.LabelFrame(canvas_cortante_1,text="Verificación
13416 resistencia a cortante del concreto [φVc]")
13416 label_cortante_check_Vs =
13417 tk.Label(labelframe_cortante_check,text="",justify="left",anchor="center",wrapleng
13418 th=400)
13417 label_cortante_check_Vs.place(x=10,y=15)
13418 labelframe_control_tamaño = tk.LabelFrame(canvas_cortante_1,text="Control del
13419 tamaño de la sección del concreto [VG]")
13419 label_control_check =
13420 tk.Label(labelframe_control_tamaño,text="",justify="left",anchor="center",wrapleng
13421 th=400)
13420 label_control_check.place(x=10,y=15)
13421 labelframe_cortante_separacion = tk.LabelFrame(canvas_cortante_1,text="Separación
13422 máxima y resistencia de estribos [S]")
13422 labelframe_cortante_check_separacion =
13423 tk.LabelFrame(canvas_cortante_1,text="Análisis de la resistencia de estribos en
13424 función de su separación")
13423 label_cortante_check_separacion =
13424 tk.Label(labelframe_cortante_check_separacion,text="",justify="left",anchor="cente
13425 r",wraplength=400)
13424 label_cortante_check_separacion.place(x=10,y=40)
13425 label_cortante_check_ramas =
13426 tk.Label(labelframe_cortante_check_separacion,text="",justify="left",anchor="cente
13427 r",wraplength=400)
13426 label_cortante_check_ramas.place(x=10,y=1)
13427 labelframe_cortante_capacidad = tk.LabelFrame(canvas_cortante_1,text="Análisis
13428 comparativo por capacidad de resistencia a cortante [φVn]")
13428 labelframe_cortante_capacidad_check =
13429 tk.LabelFrame(canvas_cortante_1,text="Verificación capacidad de resistencia a
13430 cortante [φVn]")
13429 label_cortante_capacidad_check =
13430 tk.Label(labelframe_cortante_capacidad_check,text="",justify="left",anchor="center
13431 ",wraplength=400)
13430 label_cortante_capacidad_check.place(x=10,y=15)
13431
13432

```

```
13433 # Crear un Scrollbar vertical
13434 scrollbar_y = tk.Scrollbar(frame_ventana_flexion, orient="vertical",
13435                             command=canvas_flexion.yview)
13436 scrollbar_y.place(x=565,y=1,width=20,height=495)
13437 # Vincular el Scrollbar vertical al Canvas
13438 canvas_flexion.config(yscrollcommand=scrollbar_y.set)
13439
13440 # Añadir contenido al Canvas_flexion
13441 labelframe_flexion_verificar = tk.LabelFrame(canvas_flexion,text="Análisis
13442 comparativo de diseño flexión")
13443 labelframe_flexion_check = tk.LabelFrame(canvas_flexion,text="Verificación de
13444 diseño flexión")
13445 label_check = tk.Label(labelframe_flexion_check, text="", fg="black", font=
13446 ("Arial", 8, "bold"), justify="center", anchor="center",wraplength=400 )
13447 label_check.pack(padx=10, pady=10, fill="both", expand=True)
13448 labelframe_As= tk.LabelFrame(canvas_flexion,text="Análisis del momento último por
13449 el acero requerido [As]")
13450 labelframe_As_check = tk.LabelFrame(canvas_flexion,text="Resultados de Ancho y
13451 Acero Asignado [As]")
13452 label_check_As=
13453 tk.Label(labelframe_As_check,text="",justify="center",anchor="center",wraplength=4
13454 00)
13455 label_check_As.place(x=10,y=10)
13456 label_check_info =
13457 tk.Label(labelframe_As_check,text="",justify="center",anchor="center",wraplength=5
13458 00)
13459 label_check_info.place(x=10,y=40)
13460 labelframe_flexion_Mcr = tk.LabelFrame(canvas_flexion,text="Análisis de Mcr y
13461 refuerzo de acero [As']")
13462 labelframe_check_Mcr = tk.LabelFrame(canvas_flexion,text="Resultados zona de
13463 compresión [As']")
13464 label_check_Mcr=
13465 tk.Label(labelframe_check_Mcr,text="",justify="left",anchor="center",wraplength=40
13466 0)
13467 label_check_Mcr.place(x=10,y=10)
13468 label_check_desarrollo = tk.Label(labelframe_check_Mcr,text="")
13469 label_check_desarrollo.place(x=10,y=50)
13470 labelframe_Ld_traccion = tk.LabelFrame(canvas_flexion,text="Análisis de longitud
13471 de desarrollo mínimo a tracción [Ld]")
13472 labelframe_ld_compresion = tk.LabelFrame(canvas_flexion,text="Análisis de longitud
13473 de desarrollo mínimo a compresión [Ldc]")
13474 labelframe_ganchos_ldh = tk.LabelFrame(canvas_flexion,text="Análisis de longitud
13475 de desarrollo mínimo con ganchos [Ldh]")
13476 labelframe_ganchos_check = tk.LabelFrame(canvas_flexion,text="")
13477 label_ganchos_check =
13478 tk.Label(labelframe_ganchos_check,text="",justify="center",anchor="center",wraplen
13479 gth=405)
13480 label_ganchos_check.place(x=10,y=17)
13481
13482 frame_check_on = tk.Frame(labelframe_As_check,bd=2,relief="flat")
13483 frame_check_on.place(x=450,y=10,width=50,height=50)
13484 label_icon = tk.Label(frame_check_on)
13485 label_icon.place(relwidth=1, relheight=1)
13486
13487 frame_check_on_Mcr = tk.Frame(labelframe_check_Mcr,bd=2,relief="flat")
13488 frame_check_on_Mcr.place(x=450,y=10,width=50,height=50)
13489 label_icon_Mcr = tk.Label(frame_check_on_Mcr)
13490 label_icon_Mcr.place(relwidth=1, relheight=1)
13491
```

```

13474 frame_check_on_Ld = tk.Frame(labelframe_ganchos_check,bd=2,relief="flat")
13475 frame_check_on_Ld.place(x=450,y=15,width=50,height=50)
13476 label_icon_Ld = tk.Label(frame_check_on_Ld )
13477 label_icon_Ld.place(relwidth=1, relheight=1)
13478
13479 frame_check_on_cortante = tk.Frame(labelframe_cortante_check ,bd=2,relief="flat")
13480 frame_check_on_cortante.place(x=450,y=15,width=50,height=50)
13481 label_icon_cortante = tk.Label(frame_check_on_cortante)
13482 label_icon_cortante.place(relwidth=1, relheight=1)
13483
13484 frame_check_on_cortante_1 = tk.Frame(labelframe_control_tamaño
, bd=2,relief="flat")
13485 frame_check_on_cortante_1.place(x=450,y=15,width=50,height=50)
13486 label_icon_cortante_1 = tk.Label(frame_check_on_cortante_1)
13487 label_icon_cortante_1.place(relwidth=1, relheight=1)
13488
13489 frame_check_on_cortante_2 =
tk.Frame(labelframe_cortante_check_separacion,bd=2,relief="flat")
13490 frame_check_on_cortante_2.place(x=450,y=15,width=50,height=50)
13491 label_icon_cortante_2 = tk.Label(frame_check_on_cortante_2)
13492 label_icon_cortante_2.place(relwidth=1, relheight=1)
13493
13494 frame_check_on_cortante_3 =
tk.Frame(labelframe_cortante_capacidad_check,bd=2,relief="flat")
13495 frame_check_on_cortante_3.place(x=450,y=15,width=50,height=50)
13496 label_icon_cortante_3 = tk.Label(frame_check_on_cortante_3)
13497 label_icon_cortante_3.place(relwidth=1, relheight=1)
13498
13499
13500
13501 # Redimensionar la imagen al tamaño del frame (20x20)
13502 image_on = Image.open("Iconos_proyecto/on_on_on.png")
13503 image_on = image_on.resize((30, 30), Image.Resampling.LANCZOS)
13504 icon_check_on = ImageTk.PhotoImage(image_on)
13505
13506 image_off = Image.open("Iconos_proyecto/off_off_off.png")
13507 image_off = image_off.resize((30, 30), Image.Resampling.LANCZOS)
13508 icon_check_off = ImageTk.PhotoImage(image_off)
13509
13510 "*****TREEVIEW
PRIMERO*****"
13511 # Crear el primer Treeview
13512 treeview_flexion = ttk.Treeview(labelframe_flexion_verificar, columns=
("Mmax_Mmin", "Mu_kNm", "Cuantia", "φMn_kNm", "Mu ≤ φMn"), show="headings")
13513 treeview_flexion.place(x=10,y=20,width=500,height=100)
13514 # Crear el primer Treeview de cortante
13515 treeview_cortante_1 = ttk.Treeview(labelframe_cortante_verificar, columns=("Wu",
"Vmax", "Vu", "φVc", "φVc/2", "Vs"), show="headings")
13516 treeview_cortante_1.place(x=10,y=20,width=500,height=100)
13517
13518 # Establecer los encabezados de las columnas
13519 treeview_flexion.heading("Mmax_Mmin", text="Mmax / Mmin")
13520 treeview_flexion.heading("Mu_kNm", text="Mu [kNm]")
13521 treeview_flexion.heading("Cuantia", text="Cuantia")
13522 treeview_flexion.heading("φMn_kNm", text="φMn [kNm]")
13523 treeview_flexion.heading("Mu ≤ φMn", text="Mu ≤ φMn")
13524 treeview_cortante_1.heading("Wu",text="Wu")
13525 treeview_cortante_1.heading("Vmax",text="Vmax")
13526 treeview_cortante_1.heading("Vu",text="Vu")
13527 treeview_cortante_1.heading("φVc",text="φVc")

```

```

13528 treeview_cortante_1.heading("φVc/2",text="φVc/2")
13529 treeview_cortante_1.heading("Vs",text="Vs")
13530
13531 # Modificar el ancho de las columnas
13532 treeview_flexion.column("Mmax_Mmin", width=100, stretch=False, anchor="center")
13533 treeview_flexion.column("Mu_kNm", width=100, stretch=False, anchor="center")
13534 treeview_flexion.column("Cuantia", width=100, stretch=False, anchor="center")
13535 treeview_flexion.column("φMn_kNm", width=100, stretch=False, anchor="center")
13536 treeview_flexion.column("Mu ≤ φMn", width=100, stretch=False, anchor="center")
13537 treeview_cortante_1.column("Wu", width=83, stretch=False, anchor="center")
13538 treeview_cortante_1.column("Vmax", width=83, stretch=False, anchor="center")
13539 treeview_cortante_1.column("Vu", width=83, stretch=False, anchor="center")
13540 treeview_cortante_1.column("φVc", width=83, stretch=False, anchor="center")
13541 treeview_cortante_1.column("φVc/2", width=83, stretch=False, anchor="center")
13542 treeview_cortante_1.column("Vs", width=83, stretch=False, anchor="center")
13543
13544 # Definir algunas filas de ejemplo
13545 treeview_flexion.insert("", "end", values=("Mu[-]", "", "", "", ""))
13546 treeview_flexion.insert("", "end", values=("Mu[+]", "", "", "", ""))
13547 treeview_flexion.insert("", "end", values=("Mu[-]", "", "", "", ""))
13548 treeview_cortante_1.insert("", "end", values("", "", "", "", ""))
13549
13550 #Contenido Canvas_Flexion
13551 label_flexion_a = tk.Label(canvas_flexion,text="")
13552 #Contenido canvas_cortante_1
13553 label_cortante_a = tk.Label(canvas_cortante_1,text="")
13554
13555 # Función para restaurar el tamaño de las columnas después de intentar
redimensionar
13556 def no_resize(event):
13557
13558     for col in treeview_cortante_1["columns"]:
13559         treeview_cortante_1.column(col, width=83)
13560
13561 # Vincular el evento de ButtonRelease-1 para evitar el redimensionamiento
13562 treeview_cortante_1.bind("<ButtonRelease-1>", no_resize)
13563
13564 # Función para restaurar el tamaño de las columnas después de intentar
redimensionar
13565 def no_resize(event):
13566
13567     for col in treeview_flexion["columns"]:
13568         treeview_flexion.column(col, width=100)
13569
13570 # Vincular el evento de ButtonRelease-1 para evitar el redimensionamiento
13571 treeview_flexion.bind("<ButtonRelease-1>", no_resize)
13572
13573 "*****TREEVIEW
SEGUNDO*****"
13574 # Crear el segundo Treeview
13575 treeview_refuerzo = ttk.Treeview(labelframe_As, columns=("Mu_kNm", "As_mm2",
"#Barra", "Base_requerida"), show="headings")
13576 treeview_refuerzo.place(x=10,y=20,width=500,height=100)
13577 # Crear el segundo Treeview cortante
13578 treeview_cortante_2 = ttk.Treeview(labelframe_cortante_separacion,columns=("Smax",
"Av", "#Ramas", "Vs"), show="headings")
13579 treeview_cortante_2.place(x=10,y=20,width=500,height=100)
13580
13581 # Definir los encabezados de las columnas
13582 treeview_refuerzo.heading("#1", text="Mu [kNm]")

```

```

13583 treeview_refuerzo.heading("#2", text="As [mm²]")
13584 treeview_refuerzo.heading("#3", text="#Barra")
13585 treeview_refuerzo.heading("#4", text="Base requerida")
13586 treeview_cortante_2.heading("Smax", text="Smax")
13587 treeview_cortante_2.heading("Av", text="Av")
13588 treeview_cortante_2.heading("#Ramas", text="#Ramas")
13589 treeview_cortante_2.heading("Vs", text="Vs")
13590
13591
13592 # Modificar el ancho de las columnas
13593 treeview_refuerzo.column("#1", width=100, stretch=False, anchor="center")
13594 treeview_refuerzo.column("#2", width=100, stretch=False, anchor="center")
13595 treeview_refuerzo.column("#3", width=100, stretch=False, anchor="center")
13596 treeview_refuerzo.column("#4", width=195, stretch=False, anchor="center")
13597 treeview_cortante_2.column("Smax", width=124, stretch=False, anchor="center")
13598 treeview_cortante_2.column("Av", width=124, stretch=False, anchor="center")
13599 treeview_cortante_2.column("#Ramas", width=124, stretch=False, anchor="center")
13600 treeview_cortante_2.column("Vs", width=124, stretch=False, anchor="center")
13601
13602 # Agregar las filas con la información
13603 treeview_refuerzo.insert("", "end", values=("", "", "", ""))
13604 treeview_refuerzo.insert("", "end", values=("", "", "", ""))
13605 treeview_refuerzo.insert("", "end", values=("", "", "", ""))
13606 treeview_cortante_2.insert("", "end", values=("", "", "", ""))
13607
13608 # Función para restaurar el tamaño de la columna 4 después de intentar
redimensionar
13609 def no_resize(event):
13610     # Restaurar el tamaño original solo de la columna 4
13611     treeview_cortante_2.column("Smax", width=124, stretch=False, anchor="center")
13612     treeview_cortante_2.column("Av", width=124, stretch=False, anchor="center")
13613     treeview_cortante_2.column("#Ramas", width=124, stretch=False,
anchor="center")
13614     treeview_cortante_2.column("Vs", width=124, stretch=False, anchor="center")
13615
13616 # Vincular el evento de ButtonRelease-1 para evitar el redimensionamiento
13617 treeview_cortante_2.bind("<ButtonRelease-1>", no_resize)
13618
13619 # Función para restaurar el tamaño de la columna 4 después de intentar
redimensionar
13620 def no_resize(event):
13621     # Restaurar el tamaño original solo de la columna 4
13622     treeview_refuerzo.column("#1", width=100, stretch=False, anchor="center")
13623     treeview_refuerzo.column("#2", width=100, stretch=False, anchor="center")
13624     treeview_refuerzo.column("#3", width=100, stretch=False, anchor="center")
13625     treeview_refuerzo.column("#4", width=195, stretch=False, anchor="center")
13626
13627
13628 # Vincular el evento de ButtonRelease-1 para evitar el redimensionamiento
13629 treeview_refuerzo.bind("<ButtonRelease-1>", no_resize)
13630
13631 "*****TREEVIEW
TERCERO*****"
13632 # Crear el Tercero Treeview
13633 treeview_Mcr = ttk.Treeview(labelframe_flexion_Mcr, columns=("Viga", "Mcr",
"pmin", "As'", "#Barras"), show="headings")
13634 treeview_Mcr.place(x=10, y=20, width=500, height=100)
13635 # Crear el Tercero Treeview cortante
13636 treeview_cortante_3 = ttk.Treeview(labelframe_cortante_capacidad, columns=("Vs",
"Vu", "φVn", "Vu≤φVn"), show="headings")

```

```

13637 treeview_cortante_3.place(x=10, y=20, width=500, height=100)
13638
13639 # Definir los encabezados de las columnas
13640 treeview_Mcr.heading("Viga", text="Viga")
13641 treeview_Mcr.heading("Mcr", text="Mcr [kNm]")
13642 treeview_Mcr.heading("pmin", text="pmin")
13643 treeview_Mcr.heading("As'", text="As' [mm²]")
13644 treeview_Mcr.heading("#Barras", text="#Barra")
13645 treeview_cortante_3.heading("Vs", text="Vs")
13646 treeview_cortante_3.heading("Vu", text="Vu")
13647 treeview_cortante_3.heading("φVn", text="φVn")
13648 treeview_cortante_3.heading("Vu≤φVn", text="Vu≤φVn")
13649
13650 # Modificar el ancho de las columnas
13651 treeview_Mcr.column("Viga", width=100, stretch=False, anchor="center")
13652 treeview_Mcr.column("Mcr", width=100, stretch=False, anchor="center")
13653 treeview_Mcr.column("pmin", width=100, stretch=False, anchor="center")
13654 treeview_Mcr.column("As'", width=100, stretch=False, anchor="center")
13655 treeview_Mcr.column("#Barras", width=100, stretch=False, anchor="center")
13656 treeview_cortante_3.column("Vs", width=124, stretch=False, anchor="center")
13657 treeview_cortante_3.column("Vu", width=124, stretch=False, anchor="center")
13658 treeview_cortante_3.column("φVn", width=124, stretch=False, anchor="center")
13659 treeview_cortante_3.column("Vu≤φVn", width=124, stretch=False, anchor="center")
13660
13661 # Insertar filas en el Treeview
13662 treeview_Mcr.insert("", "end", values=("", "", "", "", ""))
13663 treeview_Mcr.insert("", "end", values=("", "", "", "", ""))
13664 treeview_Mcr.insert("", "end", values=("", "", "", "", ""))
13665 treeview_cortante_3.insert("", "end", values=("", "", "", ""))
13666
13667 def no_resize(event):
13668     treeview_cortante_3.column("Vs", width=124, stretch=False, anchor="center")
13669     treeview_cortante_3.column("Vu", width=124, stretch=False, anchor="center")
13670     treeview_cortante_3.column("φVn", width=124, stretch=False, anchor="center")
13671     treeview_cortante_3.column("Vu≤φVn", width=124, stretch=False,
anchor="center")
13672
13673 # Vincular el evento de ButtonRelease-1 para evitar el redimensionamiento
13674 treeview_cortante_3.bind("<ButtonRelease-1>", no_resize)
13675
13676 # Función para restaurar el tamaño de la columna 4 después de intentar
redimensionar
13677 def no_resize(event):
13678     treeview_Mcr.column("Viga", width=100, stretch=False, anchor="center")
13679     treeview_Mcr.column("Mcr", width=100, stretch=False, anchor="center")
13680     treeview_Mcr.column("pmin", width=100, stretch=False, anchor="center")
13681     treeview_Mcr.column("As'", width=100, stretch=False, anchor="center")
13682     treeview_Mcr.column("#Barras", width=100, stretch=False, anchor="center")
13683
13684 # Vincular el evento de ButtonRelease-1 para evitar el redimensionamiento
13685 treeview_Mcr.bind("<ButtonRelease-1>", no_resize)
13686
13687 "*****TREEVIEW
CUARTO*****"
13688 # Crear el cuarto Treeview
13689 treeview_Ld_1 = ttk.Treeview(labelframe_Ld_traccion, columns=("As", "Barra",
"φBarra", "Ld"), show="headings")
13690 treeview_Ld_1.place(x=10, y=20, width=500, height=100)
13691
13692 # Definir los encabezados de las columnas

```

```

13693 treeview_Ld_1.heading("As", text="As")
13694 treeview_Ld_1.heading("Barra", text="#Barra")
13695 treeview_Ld_1.heading("ØBarra", text="ØBarra [mm]")
13696 treeview_Ld_1.heading("Ld", text="Ld [cm]")
13697
13698 # Modificar el ancho de las columnas
13699 treeview_Ld_1.column("As", width=124, stretch=False, anchor="center")
13700 treeview_Ld_1.column("Barra", width=124, stretch=False, anchor="center")
13701 treeview_Ld_1.column("ØBarra", width=124, stretch=False, anchor="center")
13702 treeview_Ld_1.column("Ld", width=125, stretch=False, anchor="center")
13703
13704 # Insertar filas iniciales en el Treeview con identificadores únicos
13705 fila1_id = treeview_Ld_1.insert("", "end", values=("As[-]", "", "", ""))
13706 fila2_id = treeview_Ld_1.insert("", "end", values=("As[+]", "", "", ""))
13707 fila3_id = treeview_Ld_1.insert("", "end", values=("As[-]", "", "", ""))
13708
13709 # Función para restaurar el tamaño de las columnas después de intentar
redimensionar
13710 def no_resize(event):
13711     treeview_Ld_1.column("As", width=124, stretch=False, anchor="center")
13712     treeview_Ld_1.column("Barra", width=124, stretch=False, anchor="center")
13713     treeview_Ld_1.column("ØBarra", width=124, stretch=False, anchor="center")
13714     treeview_Ld_1.column("Ld", width=124, stretch=False, anchor="center")
13715
13716 # Vincular el evento de ButtonRelease-1 para evitar el redimensionamiento
13717 treeview_Ld_1.bind("<ButtonRelease-1>", no_resize)
13718
13719 "*****TREEVIEW
QUINTO*****"
13720 # Crear el quinto Treeview
13721 treeview_Ldc = ttk.Treeview(labelframe_ld_compresion, columns=("As", "Barra",
"ØBarra", "Ldc"), show="headings")
13722 treeview_Ldc.place(x=10, y=20, width=500, height=100)
13723
13724 # Definir los encabezados de las columnas
13725 treeview_Ldc.heading("As", text="As")
13726 treeview_Ldc.heading("Barra", text="#Barra")
13727 treeview_Ldc.heading("ØBarra", text="ØBarra [mm]")
13728 treeview_Ldc.heading("Ldc", text="Ldc [cm]")
13729
13730 # Modificar el ancho de las columnas
13731 treeview_Ldc.column("As", width=124, stretch=False, anchor="center")
13732 treeview_Ldc.column("Barra", width=124, stretch=False, anchor="center")
13733 treeview_Ldc.column("ØBarra", width=124, stretch=False, anchor="center")
13734 treeview_Ldc.column("Ldc", width=124, stretch=False, anchor="center")
13735
13736 # Insertar filas iniciales en el Treeview con identificadores únicos
13737 fila1_ldc = treeview_Ldc.insert("", "end", values=("As'", "", "", ""))
13738
13739 # Función para restaurar el tamaño de las columnas después de intentar
redimensionar
13740 def no_resize(event):
13741     treeview_Ldc.column("As", width=124, stretch=False, anchor="center")
13742     treeview_Ldc.column("Barra", width=124, stretch=False, anchor="center")
13743     treeview_Ldc.column("ØBarra", width=124, stretch=False, anchor="center")
13744     treeview_Ldc.column("Ldc", width=124, stretch=False, anchor="center")
13745
13746 # Vincular el evento de ButtonRelease-1 para evitar el redimensionamiento
13747 treeview_Ldc.bind("<ButtonRelease-1>", no_resize)
13748

```

```

13749 """*****TREEVIEW
SEXTO*****"""
13750 # Crear el sexto Treeview
13751 treeview_ganchos_Ldh = ttk.Treeview(labelframe_ganchos_ldh, columns=("Zona",
"Barras", "Ldh", "Gancho", "Longitud"), show="headings")
13752 treeview_ganchos_Ldh.place(x=10, y=20, width=500, height=100)
13753
13754 # Definir los encabezados de las columnas
13755 treeview_ganchos_Ldh.heading("Zona", text="Zona")
13756 treeview_ganchos_Ldh.heading("Barras", text="#Barra")
13757 treeview_ganchos_Ldh.heading("Ldh", text="Ldh [cm]")
13758 treeview_ganchos_Ldh.heading("Gancho", text="Gancho")
13759 treeview_ganchos_Ldh.heading("Longitud", text="Longitud [cm]")
13760
13761 # Modificar el ancho de las columnas
13762 treeview_ganchos_Ldh.column("Zona", width=99, stretch=False, anchor="center")
13763 treeview_ganchos_Ldh.column("Barras", width=99, stretch=False, anchor="center")
13764 treeview_ganchos_Ldh.column("Ldh", width=99, stretch=False, anchor="center")
13765 treeview_ganchos_Ldh.column("Gancho", width=99, stretch=False, anchor="center")
13766 treeview_ganchos_Ldh.column("Longitud", width=99, stretch=False, anchor="center")
13767
13768 # Insertar filas iniciales en el Treeview con identificadores únicos
13769 fila_gancho_1 = treeview_ganchos_Ldh.insert("", "end", values=("Tracción", "", "",
"", ""))
13770 fila_gancho_2 = treeview_ganchos_Ldh.insert("", "end", values=("Compresión", "",
"", "", ""))
13771
13772 # Función para restaurar el tamaño de las columnas después de intentar
redimensionar
13773 def no_resize(event):
13774     treeview_ganchos_Ldh.column("Zona", width=99, stretch=False, anchor="center")
13775     treeview_ganchos_Ldh.column("Barras", width=99, stretch=False,
anchor="center")
13776     treeview_ganchos_Ldh.column("Ldh", width=99, stretch=False, anchor="center")
13777     treeview_ganchos_Ldh.column("Gancho", width=99, stretch=False,
anchor="center")
13778     treeview_ganchos_Ldh.column("Longitud", width=99, stretch=False,
anchor="center")
13779
13780 # Vincular el evento de ButtonRelease-1 para evitar el redimensionamiento
13781 treeview_ganchos_Ldh.bind("<ButtonRelease-1>", no_resize)
13782
13783 # Propiedad para insertar contenido en el canvas_flexion
13784 canvas_flexion.create_window(100,300,window=label_flexion_a)
13785 canvas_flexion.create_window(375,120,window=labelframe_flexion_verificar,width=525
,height=170)
13786 canvas_flexion.create_window(375,270,window=labelframe_flexion_check,width=525,hei
ght=100)
13787 canvas_flexion.create_window(375,420,window=labelframe_As,width=525,height=170)
13788 canvas_flexion.create_window(375,570,window=labelframe_As_check,width=525,height=1
00)
13789 canvas_flexion.create_window(375,720,window=labelframe_flexion_Mcr,width=525,heigh
t=170)
13790 canvas_flexion.create_window(375,870,window=labelframe_check_Mcr,width=525,height=
100)
13791 canvas_flexion.create_window(375,1020,window=labelframe_Ld_traccion,width=525,heig
ht=170)
13792 canvas_flexion.create_window(375,1200,window=labelframe_ld_compresion,width=525,he
ight=170)

```

```
13793 canvas_flexion.create_window(375,1380>window=labelframe_ganchos_ldh,width=525,height=170)
13794 canvas_flexion.create_window(375,1530>window=labelframe_ganchos_check,width=525,height=100)
13795
13796 # Propiedad par insertar contenido en el canvas_cortante_1
13797 canvas_cortante_1.create_window(100,300>window=label_cortante_a)
13798 canvas_cortante_1.create_window(375,120>window=labelframe_cortante_verificar,width=525,height=170)
13799 canvas_cortante_1.create_window(375,270>window=labelframe_cortante_check,width=525,height=100)
13800 canvas_cortante_1.create_window(375,380>window=labelframe_control_tamaño,width=525,height=100)
13801 canvas_cortante_1.create_window(375,530>window=labelframe_cortante_separacion,width=525,height=170)
13802 canvas_cortante_1.create_window(375,680>window=labelframe_cortante_check_separacion,width=525,height=100)
13803 canvas_cortante_1.create_window(375,830>window=labelframe_cortante_capacidad,width=525,height=170)
13804 canvas_cortante_1.create_window(375,980>window=labelframe_cortante_capacidad_check,width=525,height=100)
13805
13806 # Ajustar el tamaño del canvas y el área de desplazamiento
13807 def update_scrollregion(event=None):
13808     bbox = canvas_cortante_1.bbox("all")
13809     if bbox:
13810         # Añadir un margen adicional de 100 píxeles a las coordenadas del área
13811         canvas_cortante_1.config(scrollregion=(0, 0, bbox[2] + 500, bbox[3] + 50))
13812
13813 # Llamar a update_scrollregion para ajustar el área de desplazamiento
13814 canvas_cortante_1.bind("<Configure>", update_scrollregion)
13815
13816 # Hacer que el contenido del canvas sea desplazable
13817 canvas_cortante_1.config(scrollregion=canvas_cortante_1.bbox("all"))
13818
13819 # Ajustar el tamaño del canvas y el área de desplazamiento
13820 def update_scrollregion(event=None):
13821     bbox = canvas_flexion.bbox("all")
13822     if bbox:
13823         # Añadir un margen adicional de 100 píxeles a las coordenadas del área
13824         canvas_flexion.config(scrollregion=(0, 0, bbox[2] + 500, bbox[3] + 50))
13825
13826 # Llamar a update_scrollregion para ajustar el área de desplazamiento
13827 canvas_flexion.bind("<Configure>", update_scrollregion)
13828
13829 # Hacer que el contenido del canvas sea desplazable
13830 canvas_flexion.config(scrollregion=canvas_flexion.bbox("all"))
13831
13832 # Empacar el Notebook dentro de frame_interno
13833 notebook_flexion.pack(fill="both", expand=True)
13834
13835 # Función para actualizar cuando se cambia el valor en entry_flexion_recubrimiento
13836 def actualizar_profundidad(event):
13837     # Llamar a actualizar dimensiones para recalcular Altura_d cuando
13838     # recubrimiento cambie
13839     actualizar_dimensiones(event)
13840
13841 # Variable para controlar el estado de los radiobuttons (marca inicial)
13842 estado_estribo = tk.StringVar()
```

```
13843 # Función para actualizar el label del estribo y el área
13844 def actualizar_label_estribo(valor):
13845     # Actualizar el texto en el label_prueba_estribo
13846     label_prueba_estribo.config(text=f"{valor} mm")
13847
13848     # Cambiar el valor de la variable para asegurar que solo uno esté seleccionado
13849     estado_estribo.set(valor)
13850
13851     # Determinar el tipo de estribo y el área correspondiente
13852     if valor == 9.5:
13853         tipo_estribo = "E#3"
13854         area_estribo = 71 # Área para E#3
13855     elif valor == 12.7:
13856         tipo_estribo = "E#4"
13857         area_estribo = 129 # Área para E#4
13858     else:
13859         tipo_estribo = "Desconocido"
13860         area_estribo = 0
13861
13862     # Actualizar los labels con el tipo de estribo y el área
13863     label_estribo.config(text=f"{tipo_estribo}")
13864     label_area_estribo.config(text=f"{area_estribo} mm²")
13865
13866 label_prueba_estribo= tk.Label(ventana,text="",bg="red")
13867 #label_prueba_estribo.place(x=500,y=680,width=100)
13868
13869 frame_ventana_viga_1 = tk.Frame(ventana, bd=10, relief="ridge", bg="white")
13870 frame_ventana_viga_1.place(x=437, y=405, width=400, height=370)
13871
13872 # Crear un Label dentro del frame para mostrar la imagen
13873 label_imagen = tk.Label(frame_ventana_viga_1, bg="white")
13874 label_imagen.pack(fill="both", expand=True)
13875
13876 label_titulo_viga_1 = tk.Label(frame_ventana_viga_1,text="Detalle armado de
viga",bd=2,relief="raise")
13877 label_titulo_viga_1.place(x=1,y=1,width=380,height=35)
13878
13879 label_subtitulo_viga_1 = tk.Frame(frame_ventana_viga_1,bd=2,relief="raise")
13880 label_subtitulo_viga_1.place(x=1,y=315,width=380,height=35)
13881
13882
13883 # Menú "Definir" con opciones
13884 menu_seccion_transversal = tk.Menu(menu_definir, tearoff=0)
13885 menu_definir.add_command(label="Materiales", image=icono_materiales,
compound="left",command=abrir_ventana_materiales)
13886 menu_seccion_transversal.add_command(label="Propiedades geométricas de la
sección", image=icono_seccion_viga,
compound="left",command=abrir_ventana_seccion_transversal)
13887 menu_definir.add_cascade(label="Sección transversal",
menu=menu_seccion_transversal, image=icono_seccion_transversal, compound="left")
13888 menu_bar.add_cascade(label="Definir", menu=menu_definir)
13889
13890 # Menú "Asignar"
13891 menu_asignar = tk.Menu(menu_bar, tearoff=0)
13892 menu_bar.add_cascade(label="Asignar", menu=menu_asignar)
13893 menu_asignar.add_command(label="Apoyos", image=icono_asignar_apoyos,compound="left"
,command=abrir_ventana_apoyos)
13894 menu_asignar.add_command(label="Cargas", image=icono_asignar_cargas,compound="left"
,command=abrir_ventana_cargas)
13895
```

```
13896 # Menú "Gráficos"
13897 menu_graficos = tk.Menu(menu_bar, tearoff=0)
13898 menu_bar.add_cascade(label="Gráficos", menu=menu_graficos)
13899 menu_graficos.add_command(label="Diagramas V &
M", image=icono_graficos, compound="left", command=abrir_ventana_graficos)
13900 menu_graficos.add_command(label="Panel de
modelado", image=icono_model_3D, compound="left", command=Abrir_ventana_Detalle_Struc
tural)
13901
13902 # Menú "Diseño Flexión"
13903 menu_diseño_flexion = tk.Menu(menu_bar, tearoff=0)
13904 menu_bar.add_cascade(label="Diseño Flexión", menu=menu_diseño_flexion)
13905 # Crear un submenú para "Concreto reforzado" y agregar la opción "Cálculos
previos"
13906 submenu_calculos = tk.Menu(menu_diseño_flexion, tearoff=0)
13907 submenu_calculos.add_command(label="Cálculos
previos", image=icono_calculos, compound="left", command=abrir_ventana_flexion)
13908 # Agregar "Concreto reforzado" con el submenú "Cálculos previos"
13909 menu_diseño_flexion.add_cascade(label="Concreto reforzado", image=icono_flexion,
compound="left", menu=submenu_calculos)
13910 menu_diseño_flexion.add_cascade(label="Barras de
refuerzo", image=icono_barras, compound="left", command=abrir_ventana_barras_refuerzo
)
13911 menu_diseño_flexion.add_cascade(label="Calcular
Mcr", image=icono_VG_AS, compound="left", command=abrir_ventana_Mcr)
13912 menu_diseño_flexion.add_cascade(label="Longitud
desarrollo", image=icono_longitud_desarrollo, compound="left", command=abrir_ventana_
longitud_desarrollo)
13913
13914 # Menú "Diseño Cortante"
13915 menu_diseño_cortante = tk.Menu(menu_bar, tearoff=0)
13916 menu_bar.add_cascade(label="Diseño Cortante", menu=menu_diseño_cortante)
13917 menu_diseño_cortante.add_command(label="Diseño
cortante", image=icono_cortante, compound="left", command=abrir_ventana_cortante)
13918
13919 # Menú "Reportes"
13920 menu_resportes_docs = tk.Menu(menu_bar, tearoff=0)
13921 menu_bar.add_cascade(label="Reportes", menu=menu_resportes_docs)
13922 menu_resportes_docs.add_command(label="Reporte - Diseño
flexión", image=icono_Doc_word, compound="left", command=Generar_reportes)
13923 menu_resportes_docs.add_command(label="Reporte - Diseño
Cortante", image=icono_Doc_word_cortante, compound="left", command=generar_reporte_co
rtante)
13924
13925 # Menú "Acerca del programa Pilot Structural"
13926 menu_acerca_pilot = tk.Menu(menu_bar, tearoff=0)
13927 menu_bar.add_cascade(label="Acerca de Pilot Structural", menu=menu_acerca_pilot)
13928 menu_acerca_pilot.add_command(label="Sobre el
programa", image=icono_pilot_structural, compound="left", command=Abrir_ventana_Pilot
_Structural)
13929
13930 # Mostrar la barra de menús
13931 ventana.config(menu=menu_bar)
13932
13933 # -----
13934 # Crear la barra de herramientas (Toolbar)
13935 # -----
13936
13937 # Crear un frame para la barra de herramientas
13938 toolbar = tk.Frame(ventana, bd=2, relief=tk.RAISED)
```

```
13939 # Añadir la barra de herramientas a la ventana
13940 toolbar.pack(side=tk.TOP, fill=tk.X)
13941
13942 # Crear un frame para la barra de herramientas izquierda
13943 toolbar_1 = tk.Frame(ventana, bd=2, relief=tk.RAISED) # Barra de herramientas con
un ancho fijo
13944 toolbar_1.place(x=1,y=35,width=35,height=738)
13945
13946 # Crear un frame para la barra de herramientas derecha
13947 toolbar_2 = tk.Frame(ventana, bd=2, relief=tk.RAISED) # Barra de herramientas con
un ancho fijo
13948 toolbar_2.place(x=1500,y=35,width=35,height=738)
13949
13950 # Crear los botones de la barra de herramientas toolbar
13951 boton_nuevo_modelo = tk.Button(toolbar, image=icono_modelo_nuevo,
compound="top",command=lambda: abrir_ventana_nueva())
13952 boton_nuevo_modelo.pack(side=tk.LEFT, padx=2, pady=2)
13953 boton_abrir_archivo =
tk.Button(toolbar,image=icono_abrir_archivo,compound="top",command=abrir_archivo)
13954 boton_abrir_archivo.pack(side=tk.LEFT, padx=2, pady=2)
13955 boton_guardar =
tk.Button(toolbar,image=icono_guardar,compound="top",command=guardar_archivo)
13956 boton_guardar.pack(side=tk.LEFT, padx=2, pady=2)
13957 boton_materiales = tk.Button(toolbar, image=icono_materiales,
compound="top",command=abrir_ventana_materiales)
13958 boton_materiales.pack(side=tk.LEFT, padx=2, pady=2)
13959 boton_seccion_viga = tk.Button(toolbar, image=icono_seccion_viga,
compound="top",command=abrir_ventana_seccion_transversal)
13960 boton_seccion_viga.pack(side=tk.LEFT, padx=2, pady=2)
13961 boton_apoyos =
tk.Button(toolbar,image=icono_asignar_apoyos,compound="top",command=abrir_ventana_
apoyos)
13962 boton_apoyos.pack(side=tk.LEFT, padx=2, pady=2)
13963 boton_cargas =
tk.Button(toolbar,image=icono_asignar_cargas,compound="top",command=abrir_ventana_
cargas)
13964 boton_cargas.pack(side=tk.LEFT, padx=2, pady=2)
13965 boton_graficas=
tk.Button(toolbar,image=icono_graficos,compound="top",command=abrir_ventana_grafic
os)
13966 boton_graficas.pack(side=tk.LEFT, padx=2, pady=2)
13967 # Crear los botones de la barra de herramientas toolbar_1
13968 boton_calculos_previos =
tk.Button(toolbar_1,image=icono_calculos,compound="top",command=abrir_ventana_flex
ion)
13969 boton_calculos_previos.pack(side=tk.TOP, padx=2, pady=2, anchor="n")
13970 boton_barras_refuerzo =
tk.Button(toolbar_1,image=icono_barras,compound="top",command=abrir_ventana_barras
_refuerzo)
13971 boton_barras_refuerzo.pack(side=tk.TOP, padx=2, pady=2, anchor="n")
13972 boton_calculo_Mcr =
tk.Button(toolbar_1,image=icono_VG_AS,compound="top",command=abrir_ventana_Mcr)
13973 boton_calculo_Mcr.pack(side=tk.TOP, padx=2, pady=2, anchor="n")
13974 boton_ld=
tk.Button(toolbar_1,image=icono_longitud_desarrollo,compound="top",command=abrir_v
entana_longitud_desarrollo)
13975 boton_ld.pack(side=tk.TOP, padx=2, pady=2, anchor="n")
13976 boton_cortante =
tk.Button(toolbar_1,image=icono_cortante,compound="top",command=abrir_ventana_cort
ante)
```

```
13977 boton_cortante.pack(side=tk.TOP,padx=2,pady=2,anchor="n")
13978 # Crear los botones de la barra de herramientas toolbar_2
13979 boton_info_programa =
    tk.Button(toolbar_2,image=icono_pilot_structural,compound="top",command=Abrir_vent
    ana_Pilot_Structural)
13980 boton_info_programa.pack(side=tk.TOP,padx=2,pady=2,anchor="n")
13981 boton_doc_word =
    tk.Button(toolbar_2,image=icono_Doc_word,compound="top",command=Generar_reportes)
13982 boton_doc_word.pack(side=tk.TOP,padx=2,pady=2,anchor="n")
13983 boton_doc_word_cortante =
    tk.Button(toolbar_2,image=icono_Doc_word_cortante,compound="top",command=generar_r
    eporte_cortante)
13984 boton_doc_word_cortante.pack(side=tk.TOP,padx=2,pady=2,anchor="n")
13985 boton_model_3D =
    tk.Button(toolbar_2,image=icono_model_3D,compound="top",command=Abrir_ventana_Deta
    lle_Structural)
13986 boton_model_3D.pack(side=tk.TOP,padx=2,pady=2,anchor="n")
13987
13988 # Ocultar la ventana principal mientras se muestra el splash
13989 ventana.withdraw()
13990
13991 # Mostrar el splash screen antes de la ventana principal
13992 mostrar_splash()
13993
13994 # Mostrar la ventana
13995 ventana.mainloop()
```