



## Manual Técnico

### Librerías Utilizadas



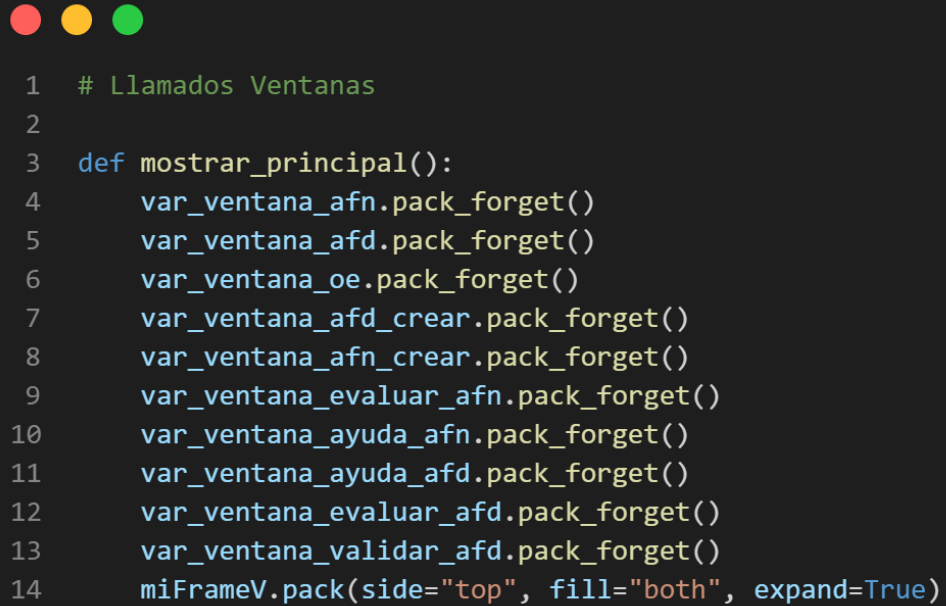
```
1 import functools
2 from tkinter import Label, Button, Frame, messagebox, Entry, StringVar, Text, ttk
3 import easygui as eg
4 from ttkthemes import ThemedTk
5 from graphviz import Digraph
6 import webbrowser
7 from reportlab.pdfgen import canvas
8 from reportlab.lib.pagesizes import A4
9 from PIL import ImageTk, Image
10 import os
```

### Variables globales de carga masiva




```
1 informacion_afn = []
2 informacion_afd = []
```

Todas las funciones de este apartado sirven, para llamar a las ventanas que se utilizaran y colocarlas en un frame variable.



```
1  # Llamados Ventanas
2
3  def mostrar_principal():
4      var_ventana_afn.pack_forget()
5      var_ventana_afd.pack_forget()
6      var_ventana_oe.pack_forget()
7      var_ventana_afd_crear.pack_forget()
8      var_ventana_afn_crear.pack_forget()
9      var_ventana_evaluar_afn.pack_forget()
10     var_ventana_ayuda_afn.pack_forget()
11     var_ventana_ayuda_afd.pack_forget()
12     var_ventana_evaluar_afd.pack_forget()
13     var_ventana_validar_afd.pack_forget()
14     miFrameV.pack(side="top", fill="both", expand=True)
```

Funcion para salir del programa.



```
1  # Funcion Salir
2
3
4  def salir():
5      exit()
```

Función que para agregar los AFN agregados manualmente en el apartado de AFN del programa.

```
1 def aceptar_afn():
2     diccionario = {}
3     transiciones_afn = campo_transiciones_afn.get('1.0','end')
4     temporal_sin_vacios = []
5     temporal = transiciones_afn.split('\n')
6     for datos in temporal:
7         if datos != '' and datos != ' ':
8             temporal_sin_vacios.append(datos)
9     transiciones_afn = temporal_sin_vacios
10    trans_corregidas = []
11    for datos in transiciones_afn:
12        lista_trans = []
13        dato = datos.split(',')
14        temporal = dato[1].split(';')
15        lista_trans.append(dato[0])
16        lista_trans.append(temporal[0])
17        lista_trans.append(temporal[1])
18        trans_corregidas.append(lista_trans)
19    diccionario["nombre"] = nombre_afn.get()
20    diccionario["estados"] = estados_afn.get().split(',')
21    diccionario["alfabeto"] = alfabeto_afn.get().split(',')
22    diccionario["estado inicial"] = estado_inicial_afn.get()
23    diccionario["estados de aceptacion"] = estados_aceptacion_afn.get().split(',')
24    diccionario["transiciones"] = trans_corregidas
25    informacion_afn.append(diccionario)
26    imprimir = '*****'
27    imprimir1 = 'AFN Agregado exitosamente'
28    mensaje = '\n\n\n\n' + imprimir.center(75, ' ') + '\n' + imprimir1.center(75, ' ') + '\n' + imprimir.center(75, ' ') + '\n'
29    messagebox.showinfo(message=mensaje, title="Mensaje")
```

Función que para agregar los AFD agregados manualmente en el apartado de AFD del programa.

```
1 def aceptar_afd():
2     diccionario = {}
3     transiciones_afd = campo_transiciones_afd.get('1.0','end')
4     temporal_sin_vacios = []
5     temporal = transiciones_afd.split('\n')
6     for datos in temporal:
7         if datos != '' and datos != ' ':
8             temporal_sin_vacios.append(datos)
9     transiciones_afd = temporal_sin_vacios
10    trans_corregidas = []
11    for datos in transiciones_afd:
12        lista_trans = []
13        dato = datos.split(',')
14        temporal = dato[1].split(';')
15        lista_trans.append(dato[0])
16        lista_trans.append(temporal[0])
17        lista_trans.append(temporal[1])
18        trans_corregidas.append(lista_trans)
19    diccionario["nombre"] = nombre_afd.get()
20    diccionario["estados"] = estados_afd.get().split(',')
21    diccionario["alfabeto"] = alfabeto_afd.get().split(',')
22    diccionario["estado inicial"] = estado_inicial_afd.get()
23    diccionario["estados de aceptacion"] = estados_aceptacion_afd.get().split(',')
24    diccionario["transiciones"] = trans_corregidas
25    informacion_afd.append(diccionario)
26    imprimir = '*****'
27    imprimir1 = 'AFD Agregado exitosamente'
28    mensaje = '\n\n\n\n' + imprimir.center(75, ' ') + '\n' + imprimir1.center(75, ' ') + '\n' + imprimir.center(75, ' ') + '\n'
29    messagebox.showinfo(message=mensaje, title="Mensaje")
```

## Función carga masiva del apartado AFD

```
1 #Carga Masiva
2 def cargar_AFD():
3     global informacion_afd
4     lista = []
5     imprimir = '*****'
6     imprimir1 = 'Archivo Cargado exitosamente'
7     imprimir2 = 'Archivo no seleccionado, vuelva a intentarlo'
8     extension = [".py", ".pyc"]
9     archivo = eg.fileopenbox(msg="Abrir archivo",
10                             title="Control: fileopenbox",
11                             default="C:/Users/neri/OneDrive/escritorio/afd",
12                             filetype=extension)
13     mensaje = 'Ruta del Archivo: ' + str(archivo) + '\n\n\n\n' + imprimir.center(
14         75, ' ') + '\n' + imprimir1.center(75, ' ') + '\n' + imprimir2.center(75, ' ') + '\n'
15     messagebox.showinfo(message=mensaje, title="Mensaje")
16     f = open(archivo, 'r', encoding="utf8")
17     leer = f.read()
18     lista_temporal = leer.split('\n')
19     for element in lista_temporal:
20         temporal_sin_vacios = ''
21         if element != '' and element != ' ':
22             temporal_sin_vacios = element
23         temporal = temporal_sin_vacios.split('\n')
24         temporal_sin_vacios = []
25         for datos in temporal:
26             if datos != '' and datos != ' ':
27                 temporal_sin_vacios.append(datos)
28         if element != '' and element != ' ':
29             lista.append(temporal_sin_vacios)
30     for element in lista:
31         diccionario = {}
32         diccionario["nombre"] = element[0]
33         diccionario["estados"] = element[1].split(',')
34         diccionario["alfabeto"] = element[2].split(',')
35         diccionario["estado inicial"] = element[3]
36         diccionario["estados de aceptacion"] = element[4].split(',')
37         transiciones = element[5:]
38         trans_corregidas = []
39         for datos in transiciones:
40             lista_trans = []
41             dato = datos.split(',')
42             temporal = dato[1].split(':')
43             lista_trans.append(dato[0])
44             lista_trans.append(temporal[0])
45             lista_trans.append(temporal[1])
46             trans_corregidas.append(lista_trans)
47         diccionario["transiciones"] = trans_corregidas
48         informacion_afd.append(diccionario)
49     f.close()
```

## Función carga masiva AFN

```
1 def cargar_AFN():
2     global informacion_afn
3     lista = []
4     imprimir = '*****'
5     imprimir1 = 'Archivo Cargado exitosamente'
6     imprimir2 = 'Archivo no seleccionado, vuelva a intentarlo'
7     extension = [".py", ".pyc"]
8     archivo = eg.fileopenbox(msg="Abrir archivo",
9                             title="Control: fileopenbox",
10                             default="C:/Users/neri/OneDrive/escritorio/afn",
11                             filetype=extension)
12     mensaje = 'Ruta del Archivo: ' + str(archivo) + '\n\n\n\n' + imprimir.center(
13         75, ' ') + '\n' + imprimir1.center(75, ' ') + '\n' + imprimir2.center(75, ' ') + '\n'
14     messagebox.showinfo(message=mensaje, title="Mensaje")
15     f = open(archivo, 'r', encoding="utf8")
16     leer = f.read()
17     lista_temporal = leer.split('\n')
18     for element in lista_temporal:
19         temporal_sin_vacios = ''
20         if element != '' and element != ' ':
21             temporal_sin_vacios = element
22         temporal = temporal_sin_vacios.split('\n')
23         temporal_sin_vacios = []
24         for datos in temporal:
25             if datos != '' and datos != ' ':
26                 temporal_sin_vacios.append(datos)
27         if element != '' and element != ' ':
28             lista.append(temporal_sin_vacios)
29     for element in lista:
30         diccionario = {}
31         diccionario["nombre"] = element[0]
32         diccionario["estados"] = element[1].split(',')
33         diccionario["alfabeto"] = element[2].split(',')
34         diccionario["estado inicial"] = element[3]
35         diccionario["estados de aceptacion"] = element[4].split(',')
36         transiciones = element[5:]
37         trans_corregidas = []
38         for datos in transiciones:
39             lista_trans = []
40             dato = datos.split(',')
41             temporal = dato[1].split(':')
42             lista_trans.append(dato[0])
43             lista_trans.append(temporal[0])
44             lista_trans.append(temporal[1])
45             trans_corregidas.append(lista_trans)
46         diccionario["transiciones"] = trans_corregidas
47         informacion_afn.append(diccionario)
48     f.close()
```

Función que crea el grafo de los AFD

```
1  #Generacion grafica
2  def generarDOT_afd():
3      # Nombre de la carpeta
4      carpeta_automatas = "automatas"
5
6      # Crear la carpeta si no existe
7      if not os.path.exists(carpeta_automatas):
8          os.makedirs(carpeta_automatas)
9      for element in informacion_afd:
10         index = str(informacion_afd.index(element))
11         ruta_archivo = os.path.join(carpeta_automatas, f'AFDPrueba{index}')
12         dot = Digraph('AFD', filename=ruta_archivo, format='png')
13         dot.attr(rankdir='LR', size='8,5')
14         dot.attr('node', shape='doublecircle')
15         for estadosA in element["estados de aceptacion"]:
16             dot.node(estadosA)
17         dot.attr('node', shape='circle')
18         for estados in element["estados"]:
19             dot.node(estados)
20         for trans in element["transiciones"]:
21             dot.edge(trans[0], trans[2], label=trans[1])
22         dot.render(f'automatas/AFDPrueba{index}', view=False)
```

Función que crea el grafo de los AFN

```
1  def generarDOT_afn():
2      carpeta_automatas = "automatas"
3
4      # Crear la carpeta si no existe
5      if not os.path.exists(carpeta_automatas):
6          os.makedirs(carpeta_automatas)
7      for element in informacion_afn:
8         index = str(informacion_afn.index(element))
9         ruta_archivo = os.path.join(carpeta_automatas, f'AFNPrueba{index}')
10         dot = Digraph('AFN', filename=ruta_archivo, format='png')
11         dot.attr(rankdir='LR', size='8,5')
12         dot.attr('node', shape='doublecircle')
13         for estadosA in element["estados de aceptacion"]:
14             dot.node(estadosA)
15         dot.attr('node', shape='circle')
16         for estados in element["estados"]:
17             dot.node(estados)
18         for trans in element["transiciones"]:
19             dot.edge(trans[0], trans[2], label=trans[1])
20         dot.render(f'automatas/AFNPrueba{index}', view=False)
```

## Función que genera el archivo pdf del Reporte de AFD

```
1 def reporte_afd():
2     generarDOT_afd()
3     w, h = A4
4     pdf = canvas.Canvas("Reporte_afd.pdf", pagesize=A4)
5     pdf.setTitle("Reporte de AFD")
6     for element in informacion_afd:
7         index = str(informacion_afd.index(element))
8         text = pdf.beginText(50, h - 50)
9         text.setFont("Times-Roman", 12)
10        text.textLine('Nombre: ' + element['nombre'])
11        text.textLine("Estados: " + ', '.join(element['estados']))
12        text.textLine("Alfabeto: " + ', '.join(element['alfabeto']))
13        text.textLine("Estado Inicial: " + element['estado inicial'])
14        text.textLine("Estados de Aceptacion: " + ', '.join(element['estados de aceptacion']))
15        text.textLine("Transiciones:")
16        for datos in element['transiciones']:
17            text.textLine(f'{datos[0]} , {datos[1]} ; {datos[2]}')
18        text.textLine()
19        pdf.drawText(text)
20        text = "AFD generado con Graphviz"
21        text_width = pdf.stringWidth(text, "Times-Roman", 12)
22        pdf.drawCentredString(w / 2, h - 200, text)
23
24        image_path = f'automatas/AFDPueba{index}.png'
25        image_width = 200
26        image_height = 200
27        image_x = (w - image_width) / 2
28        image_y = h - 250 - image_height # Adjust the vertical position as needed
29        pdf.drawInlineImage(image_path, image_x, image_y, width=image_width, height=image_height, preserveAspectRatio=True)
30        pdf.showPage() # Agregar una nueva página antes de la imagen
31    pdf.save()
32    webbrowser.open_new_tab('Reporte_afd.pdf')
```

## Función que genera el archivo pdf del Reporte de AFN

```
1 def reporte_afn():
2     generarDOT_afn()
3     w, h = A4
4     pdf = canvas.Canvas("Reporte_afn.pdf", pagesize=A4)
5     pdf.setTitle("Reporte de AFN")
6     for element in informacion_afn:
7         index = str(informacion_afn.index(element))
8         text = pdf.beginText(50, h - 50)
9         text.setFont("Times-Roman", 12)
10        text.textLine('Nombre: ' + element['nombre'])
11        text.textLine("Estados: " + ', '.join(element['estados']))
12        text.textLine("Alfabeto: " + ', '.join(element['alfabeto']))
13        text.textLine("Estado Inicial: " + element['estado inicial'])
14        text.textLine("Estados de Aceptacion: " + ', '.join(element['estados de aceptacion']))
15        text.textLine("Transiciones:")
16        for datos in element['transiciones']:
17            text.textLine(f'{datos[0]} , {datos[1]} ; {datos[2]}')
18        text.textLine()
19        pdf.drawText(text)
20        text = "AFN generado con Graphviz"
21        text_width = pdf.stringWidth(text, "Times-Roman", 12)
22        pdf.drawCentredString(w / 2, h - 200, text)
23
24        image_path = f'automatas/AFNPueba{index}.png'
25        image_width = 200
26        image_height = 200
27        image_x = (w - image_width) / 2
28        image_y = h - 250 - image_height # Adjust the vertical position as needed
29        pdf.drawInlineImage(image_path, image_x, image_y, width=image_width, height=image_height, preserveAspectRatio=True)
30        pdf.showPage() # Agregar una nueva página antes de la imagen
31    pdf.save()
32    webbrowser.open_new_tab('Reporte_afn.pdf')
```

Las funciones de apartado ventana sirve para tener la maquetación de las ventanas gráficas, que se agregan a el frame variable

```
1 # Ventanas
2 def ventana_afn(master, callback=None, args=(), kwargs={}):
3     if callback is not None:
4         callback = functools.partial(callback, *args, **kwargs)
5     main_frame = Frame(master)
6     # frame centrado
7     frame_centrado = Frame(main_frame, height=310, width=450)
8     frame_centrado.place(relx=0.5, rely=0.5, anchor="center")
9     # agregando botones
10    label = Label(frame_centrado, text="Modulo AFN")
11    label.grid(row=0, column=0, padx=10, pady=10)
12    boton_crear = Button(frame_centrado, text='Crear AFN',
13                        width=15, height=3, bd="4", command=mostrar_ventana_afn_crear)
14    boton_crear.grid(row=1, column=0, padx=10, pady=10)
15    boton_evaluar = Button(frame_centrado, text='Evaluar Cadena',
16                        width=15, height=3, bd="4", command=mostrar_ventana_evaluar_afn)
17    boton_evaluar.grid(row=2, column=0, padx=10, pady=10)
18    boton_reporte = Button(frame_centrado, text='Generar Reporte',
19                        width=15, height=3, bd="4", command=reporte_afn)
20    boton_reporte.grid(row=3, column=0, padx=10, pady=10)
21    boton_ayuda = Button(frame_centrado, text='Ayuda',
22                        width=15, height=3, bd="4", command=mostrar_ventana_ayuda_afn)
23    boton_ayuda.grid(row=4, column=0, padx=10, pady=10)
24    boton_regresar = Button(frame_centrado, text='Regresar', width=15,
25                        height=3, command=callback, bd="4")
26    boton_regresar.grid(row=5, column=0, padx=10, pady=10)
27    return main_frame
```

Función para validar las cadenas de los AFD

```
1 #Evaluador de cadenas
2 def verificar_cadena_en_afd(cadena, afd):
3     imprimir = "*****"
4     estado_actual = afd['estado inicial']
5     for caracter in cadena:
6         if caracter not in afd['alfabeto']:
7             imprimir1 = f'\n\n\n\nEl caracter: {caracter}\n no está en el alfabeto del AFD: {afd["nombre"]}'
8             mensaje = '\n\n\n\n' + imprimir1.center(75, ' ') + imprimir1.center(75, ' ') + '\n'
9             messagebox.showinfo(message=mensaje, title="Mensaje")
10            return False
11
12    transicion_encontrada = False
13    for transicion in afd['transiciones']:
14        if transicion[0] == estado_actual and transicion[1] == caracter:
15            estado_actual = transicion[2]
16            transicion_encontrada = True
17            break
18
19    if not transicion_encontrada:
20        imprimir1 = f"No hay una transición definida para el estado '{estado_actual}' y el caracter '{caracter}' en el AFD: '{afd["nombre"]}'"
21        mensaje = '\n\n\n\n' + imprimir1.center(75, ' ') + imprimir1.center(75, ' ') + '\n'
22        messagebox.showinfo(message=mensaje, title="Mensaje")
23        return False
24
25    if estado_actual in afd['estados de aceptacion']:
26        imprimir1 = f'La cadena {cadena} es válida en el AFD: {afd["nombre"]}'
27        mensaje = '\n\n\n\n' + imprimir1.center(75, ' ') + imprimir1.center(75, ' ') + '\n'
28        return messagebox.showinfo(message=mensaje, title="Mensaje")
29    else:
30        imprimir1 = f'La cadena {cadena} no es válida en el AFD: {afd["nombre"]}'
31        mensaje = '\n\n\n\n' + imprimir1.center(75, ' ') + imprimir1.center(75, ' ') + '\n'
32        return messagebox.showinfo(message=mensaje, title="Mensaje")
```



Función que obtiene el automata seleccionado para validar.

```
1 def validar_cadena_afd():
2     afd_seleccionado = combo.get()
3     for element in informacion_afd:
4         indice = str(informacion_afd.index(element))
5         if element['nombre'] == afd_seleccionado:
6             afd_seleccionado = int(indice)
7     verificar_cadena_en_afd(cadena.get(),informacion_afd[afd_seleccionado])
```

Parte de la pantalla principal donde esta el frame variable

```
1 # Abro venta
2 ventana = Tk(theme="ubuntu")
3 ancho_ventana = 1280
4 alto_ventana = 720
5 x_ventana = ventana.winfo_screenwidth() // 2 - ancho_ventana // 2
6 y_ventana = ventana.winfo_screenheight() // 2 - alto_ventana // 2
7 posicion = str(ancho_ventana) + "x" + str(alto_ventana) + \
8     "+" + str(x_ventana) + "+" + str(y_ventana)
9 ventana.geometry(posicion)
10 ventana.title('Proyecto 1')
11 # Frame con nombre proyecto
12 miFrame = Frame()
13 miFrame.pack(side="top", fill="x")
14 miFrame.config(width="500", height="50", relief="solid", bd="3")
15 label = Label(miFrame, text='Lenguajes Formales y de Programación | Sección: P | 201807086 | Nery Barrientos')
16 label.pack(side="top")
17 # frame Variable
18 miFrameV = Frame()
19 miFrameV.pack(fill="x")
20 miFrameV.place(x=0, y=25)
21 miFrameV.config(width=ancho_ventana, height=alto_ventana, relief="solid", bd="3")
22 # Funciones para ventanas en botones
23 var_ventana_afn = ventana.afn(ventana, mostrar_principal)
24 var_ventana_evaluar_afn = ventana.evaluar_afn(ventana,mostrar_ventana_afn)
25 var_ventana_afn_crear = ventana.afn_crear(ventana, mostrar_ventana_afn)
26 var_ventana_ayuda_afn = ventana.afn_ayuda(ventana,mostrar_ventana_afn)
27 var_ventana_afd = ventana.afd(ventana, mostrar_principal)
28 var_ventana_evaluar_afd = ventana.evaluar_afd(ventana,mostrar_ventana_afd)
29 var_ventana_validar_afd = ventana.validar_afd(ventana,mostrar_ventana_evaluar_afd)
30 var_ventana_afd_crear = ventana.afd_crear(ventana, mostrar_ventana_afd)
31 var_ventana_ayuda_afd = ventana.afd_ayuda(ventana,mostrar_ventana_afd)
32 var_ventana_oe = ventana.oe(ventana, mostrar_principal)
33 var_ventana_cargar_archivos = ventana.cargar(ventana, mostrar_principal)
34 # agregando Items a frame Variable
35 frame_centrado = Frame(miFrameV, height=310, width=450)
36 frame_centrado.place(relx=0.5, rely=0.5, anchor="center")
37 boton_afn = Button(frame_centrado, text='AFN',width=10, height=3, bd="4", command=mostrar_ventana_afn)
38 boton_afn.grid(row=0, column=0, padx=10, pady=10)
39 boton_afd = Button(frame_centrado, text='AFD',width=10, height=3, bd="4", command=mostrar_ventana_afd)
40 boton_afd.grid(row=0, column=1, padx=10, pady=10)
41 boton_oe = Button(frame_centrado, text='OE',width=10, height=3, bd="4", command=mostrar_ventana_oe)
42 boton_oe.grid(row=1, column=0, padx=10, pady=10)
43 boton_cargar = Button(frame_centrado, text='Cargar Archivo', width=10,height=3, command=mostrar_ventana_cargar, bd="4")
44 boton_cargar.grid(row=1, column=1, padx=10, pady=10)
45 boton_salir = Button(frame_centrado, text='Salir', width=10,height=3, command=salir, bd="4")
46 boton_salir.grid(row=2, column=0, padx=10, pady=10, columnspan=2)
47 # frame Inferior
48 miFrame1 = Frame()
49 miFrame1.pack(side="bottom", fill="x")
50 miFrame1.config(width="500", height="30", relief="solid", bd="3")
51
52 ventana.mainloop()
```