2022

Proyecto 2
Manual Técnico

LABORATORIO LENGUAJES FORMALES Y DE PROGRAMACION N

TOBÍAS RAFAEL ZAMORA SANTOS, 202010828

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA | 11 Avenida, Cdad. de Guatemala 01012

Índice

Introducción

El siguiente reporte describe los métodos, funciones, estructuras entre otros correspondientes al proyecto número 2, con los cuales se obtuvo el resultado esperado, un programa básico desarrollado en el lenguaje Python, que consta del ingreso de un archivo y por medio de nuestra lectura de archivos identificamos las gramáticas o autómatas de pila que vengan, según la acción que queramos hacer con los datos guardados mostrar lo que se halla seleccionado.

1. Técnica de Programación

1.1 Programación Orientada a Objetos

La programación orientada a objetos se basa en el concepto de crear un modelo del problema de destino en sus programas. La programación orientad a objetos disminuye los errores y promociona la reutilización del código, Python es un lenguaje orientado a objetos, por este motivo se decidió trabajar de este modo.

2. Convenciones de Nomenclatura

2.1 Declaración de Variables

Para la declaración de variables, se usó un nombre el cual hiciera referencia a lo que se contendrá en la variable, por ejemplo, al guardar las gramáticas que se iban ingresando, la variable correspondiente que contendrá los cursos fue nombrada “Gramaticas”.

```
self.nombre = nombre
self.alfabeto = alfabeto
self.simbolosP = simbolosP
self.estados = estados
self.estado_inicial = estado_inicial
self.estado_aceptacion = estado_aceptacion
self.transiciones = []
```

El nombre de cada variable corresponde a lo que contendrá.

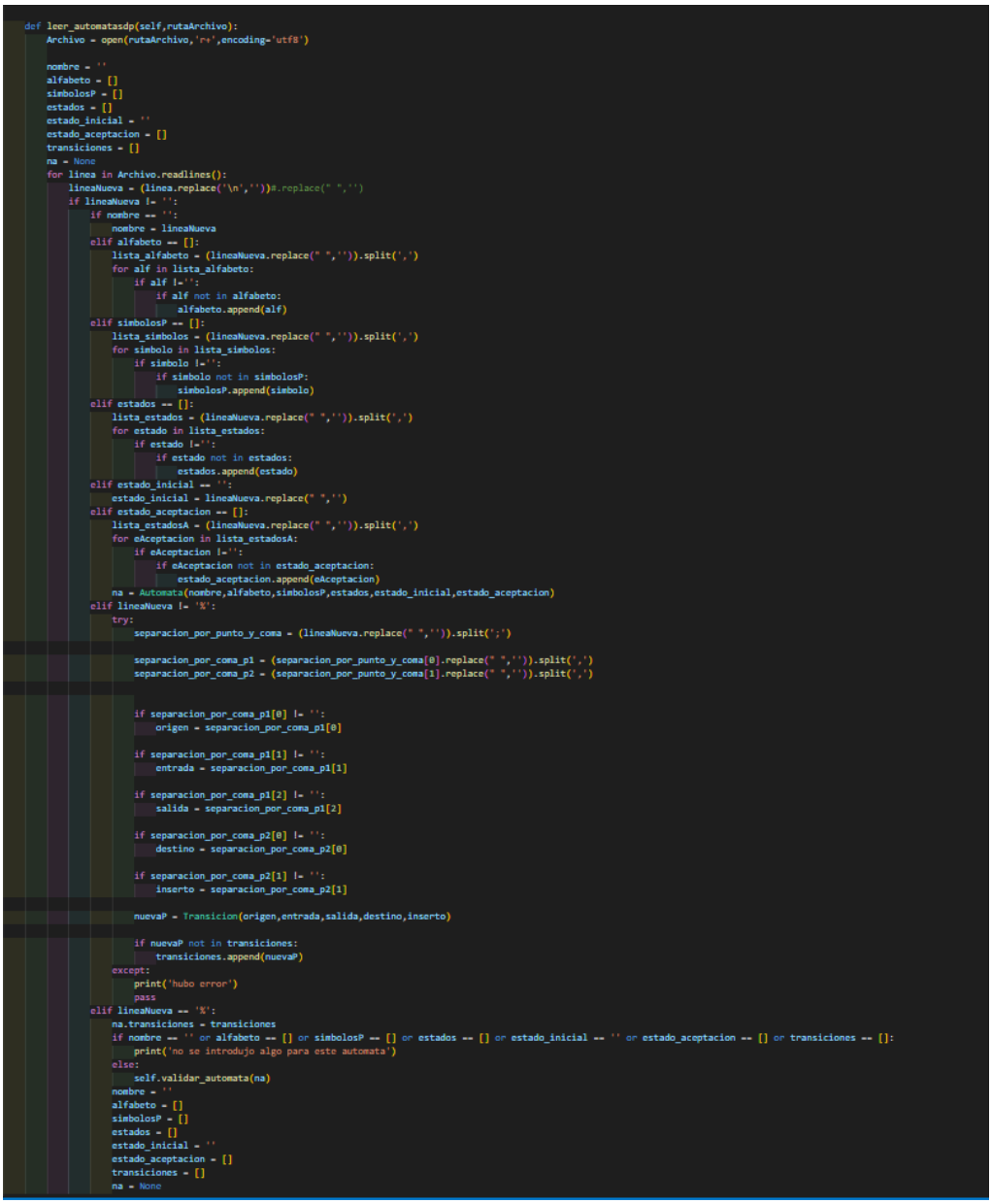
2.2 Métodos

Para la declaración de Métodos, el nombre de cada método corresponde a la acción que estos deben realizar, cabe mencionar que los métodos solo realizan las instrucciones que indiquemos, no retornan valor alguno.

Método para presentar una nueva ventana.

```
def modulo_glc(self):
    root = Toplevel()
    ModuloGLC(self.gestor,master=root,padre=self.master)
```

Método para analizar y agregar autómatas de pila



```
def leer_automatasdp(self,rutaArchivo):
    Archivo = open(rutaArchivo,'r+',encoding='utf8')

    nombre = ''
    alfabeto = []
    simbolosP = []
    estados = []
    estado_inicial = ''
    estado_aceptacion = []
    transiciones = []
    na = None
    for linea in Archivo.readlines():
        lineaNueva = (linea.replace('\n',''))#.replace(" ",'')
        if lineaNueva != '':
            if nombre == '':
                nombre = lineaNueva
            elif alfabeto == []:
                lista_alfabeto = (lineaNueva.replace(" ",'')).split(',')
                for alf in lista_alfabeto:
                    if alf !='':
                        if alf not in alfabeto:
                            alfabeto.append(alf)
            elif simbolosP == []:
                lista_simbolos = (lineaNueva.replace(" ",'')).split(',')
                for simbolo in lista_simbolos:
                    if simbolo !='':
                        if simbolo not in simbolosP:
                            simbolosP.append(simbolo)
            elif estados == []:
                lista_estados = (lineaNueva.replace(" ",'')).split(',')
                for estado in lista_estados:
                    if estado !='':
                        if estado not in estados:
                            estados.append(estado)
            elif estado_inicial == '':
                estado_inicial = lineaNueva.replace(" ",'')
            elif estado_aceptacion == []:
                lista_estadosA = (lineaNueva.replace(" ",'')).split(',')
                for eAceptacion in lista_estadosA:
                    if eAceptacion !='':
                        if eAceptacion not in estado_aceptacion:
                            estado_aceptacion.append(eAceptacion)
                na = Automata(nombre,alfabeto,simbolosP,estados,estado_inicial,estado_aceptacion)
            elif lineaNueva != '%':
                try:
                    separacion_por_punto_y_coma = (lineaNueva.replace(" ",'')).split(';')

                    separacion_por_coma_p1 = (separacion_por_punto_y_coma[0].replace(" ",'')).split(',')
                    separacion_por_coma_p2 = (separacion_por_punto_y_coma[1].replace(" ",'')).split(',')


                    if separacion_por_coma_p1[0] != '':
                        origen = separacion_por_coma_p1[0]

                    if separacion_por_coma_p1[1] != '':
                        entrada = separacion_por_coma_p1[1]

                    if separacion_por_coma_p1[2] != '':
                        salida = separacion_por_coma_p1[2]

                    if separacion_por_coma_p2[0] != '':
                        destino = separacion_por_coma_p2[0]

                    if separacion_por_coma_p2[1] != '':
                        inserto = separacion_por_coma_p2[1]

                    nuevaP = Transicion(origen,entrada,salida,destino,inserto)

                    if nuevaP not in transiciones:
                        transiciones.append(nuevaP)
                except:
                    print('hubo error')
                    pass
            elif lineaNueva == '%':
                na.transiciones = transiciones
                if nombre == '' or alfabeto == [] or simbolosP == [] or estados == [] or estado_inicial == '' or estado_aceptacion == [] or transiciones == []:
                    print('no se introdujo algo para este automata')
                else:
                    self.validar_automata(na)
                nombre = ''
                alfabeto = []
                simbolosP = []
                estados = []
                estado_inicial = ''
                estado_aceptacion = []
                transiciones = []
                na = None
```

2.3 Funciones

Para la declaración de funciones, al igual que los métodos, el nombre de cada función hace referencia a la acción que deben realizar, en este caso las funciones si retornan valores.

Función que nos regresa la lista con las gramáticas libres de contexto existentes

```
def lista_grs(self):
    return self.GramaticasLC
```

Funciones que regresan cada lista correspondiente con los datos existentes

```
def lista_atms(self):
    return self.AutomatasDPL
```

3. Métodos Principales

3.1 leer_gramaticaslc(rutaArchivo)

Este método analiza el archivo que hallamos seleccionado, lee línea por línea y separa cada parte de la gramática guardando sus valores en una variable correspondiente a los valores que se obtengan para luego crear el objeto de tipo Gramática.

```python
def leer_gramaticaslc(self,rutaArchivo):
    Archivo = open(rutaArchivo,'r+',encoding='utf8')

    nombre = ''
    no_terminales = []
    terminales = []
    no_terminal_inicial = ''
    producciones = []
    ng = None
    for linea in Archivo.readlines():
        lineaNueva = (linea.replace('\n',''))#.replace(" ",'')
        # print(lineaNueva)
        if lineaNueva != '':
            if nombre == '':
                nombre = lineaNueva
            elif no_terminales == []:
                lista_no_t = (lineaNueva.replace(" ",'')).split(',')
                for nt in lista_no_t:
                    if nt !='':
                        if nt not in no_terminales:
                            no_terminales.append(nt)
            elif terminales == []:
                lista_t = (lineaNueva.replace(" ",'')).split(',')
                for t in lista_t:
                    if t !='':
                        if t not in terminales:
                            terminales.append(t)
            elif no_terminal_inicial == '':
                no_terminal_inicial = lineaNueva.replace(" ",'')
                ng = Gramatica(nombre,lista_no_t,lista_t,no_terminal_inicial)
            elif lineaNueva != '%':
                try:
                    separacion_por_signo_mayor = lineaNueva.split('>')
                    nuevaP = Produccion(separacion_por_signo_mayor[0].replace(" ",''))
                    cont = 0
                    separacion_por_espacios = separacion_por_signo_mayor[1].split(' ')
                    while cont < len(separacion_por_espacios):
                        if separacion_por_espacios[cont] != '':
                            nuevaP.expresion.append(separacion_por_espacios[cont].replace(" ",''))
                        cont+=1
                    # while cont < len(separacion_por_signo_mayor[1]):
                    #     nuevaP.expresion.append(separacion_por_signo_mayor[1][cont])
                    #     cont+=1
                    if nuevaP not in producciones:
                        producciones.append(nuevaP)
                except:
                    pass
            elif lineaNueva == '%':
                ng.producciones = producciones
                if nombre == '' or no_terminales == [] or terminales == [] or no_terminal_inicial == '' or producciones == []:
                    print('no se introdujo algo para esta gramatica')
                else:
                    self.validar_gramatica(ng)
                nombre = ''
                no_terminales = []
                terminales = []
                no_terminal_inicial = ''
                producciones = []
                ng = None
```

3.2 leer_automatasdp(rutaArchivo)

Este método analiza el archivo que hallamos seleccionado, lee línea por línea y separa cada parte del autómata guardando sus valores en una variable correspondiente a los valores que se obtengan para luego crear el objeto de tipo Automata.

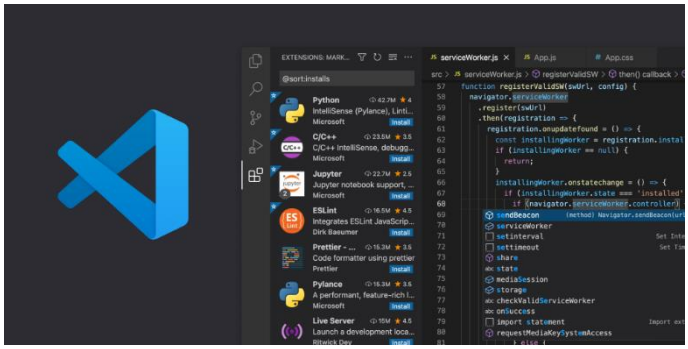
```
def leer_automatasdp(self,rutaArchivo):
    Archivo = open(rutaArchivo,'r+',encoding='utf8')

    nombre = ''
    alfabeto = []
    simbolosP = []
    estados = []
    estado_inicial = ''
    estado_aceptacion = []
    transiciones = []
    na = None
    for linea in Archivo.readlines():
        lineaNueva = (linea.replace('\n',''))#.replace(" ",'')
        if lineaNueva != '':
            if nombre == '':
                nombre = lineaNueva
            elif alfabeto == []:
                lista_alfabeto = (lineaNueva.replace(" ",'')).split(',')
                for alf in lista_alfabeto:
                    if alf !='':
                        if alf not in alfabeto:
                            alfabeto.append(alf)
            elif simbolosP == []:
                lista_simbolos = (lineaNueva.replace(" ",'')).split(',')
                for simbolo in lista_simbolos:
                    if simbolo !='':
                        if simbolo not in simbolosP:
                            simbolosP.append(simbolo)
            elif estados == []:
                lista_estados = (lineaNueva.replace(" ",'')).split(',')
                for estado in lista_estados:
                    if estado !='':
                        if estado not in estados:
                            estados.append(estado)
            elif estado_inicial == '':
                estado_inicial = lineaNueva.replace(" ",'')
            elif estado_aceptacion == []:
                lista_estadosA = (lineaNueva.replace(" ",'')).split(',')
                for eAceptacion in lista_estadosA:
                    if eAceptacion !='':
                        if eAceptacion not in estado_aceptacion:
                            estado_aceptacion.append(eAceptacion)
                na = Automata(nombre,alfabeto,simbolosP,estados,estado_inicial,estado_aceptacion)
            elif lineaNueva != '%':
                try:
                    separacion_por_punto_y_coma = (lineaNueva.replace(" ",'')).split(';')

                    separacion_por_coma_p1 = (separacion_por_punto_y_coma[0].replace(" ",'')).split(',')
                    separacion_por_coma_p2 = (separacion_por_punto_y_coma[1].replace(" ",'')).split(',')


                    if separacion_por_coma_p1[0] != '':
                        origen = separacion_por_coma_p1[0]

                    if separacion_por_coma_p1[1] != '':
                        entrada = separacion_por_coma_p1[1]

                    if separacion_por_coma_p1[2] != '':
                        salida = separacion_por_coma_p1[2]

                    if separacion_por_coma_p2[0] != '':
                        destino = separacion_por_coma_p2[0]

                    if separacion_por_coma_p2[1] != '':
                        inserto = separacion_por_coma_p2[1]

                    nuevaP = Transicion(origen,entrada,salida,destino,inserto)

                    if nuevaP not in transiciones:
                        transiciones.append(nuevaP)
                except:
                    print('hubo error')
                    pass
            elif lineaNueva == '%':
                na.transiciones = transiciones
                if nombre == '' or alfabeto == [] or simbolosP == [] or estados == [] or estado_inicial == '' or estado_aceptacion == [] or transiciones == []:
                    print('no se introdujo algo para este automata')
                else:
                    self.validar_automata(na)
                nombre = ''
                alfabeto = []
                simbolosP = []
                estados = []
                estado_inicial = ''
                estado_aceptacion = []
                transiciones = []
                na = None
```

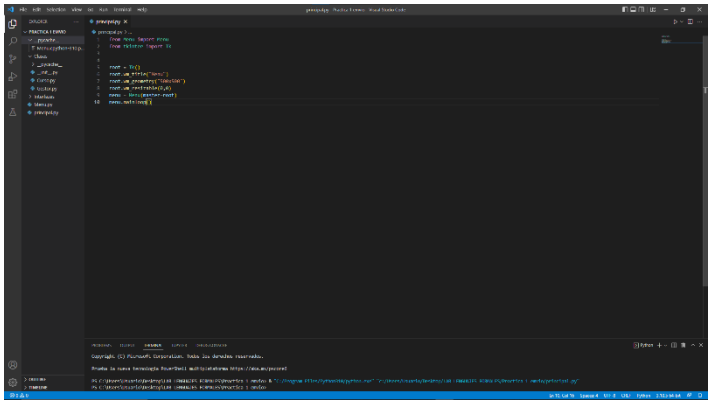
4. Requerimiento o Funcionalidad Especifica

Para la elaboración de este proyecto se usaron los siguientes programas y herramientas:

4.1 Visual Studio Code

Se uso este editor de código debido a que ya se tenía conocimiento sobre él, por este motivo fue más sencillo llevar a cabo el proyecto en él.





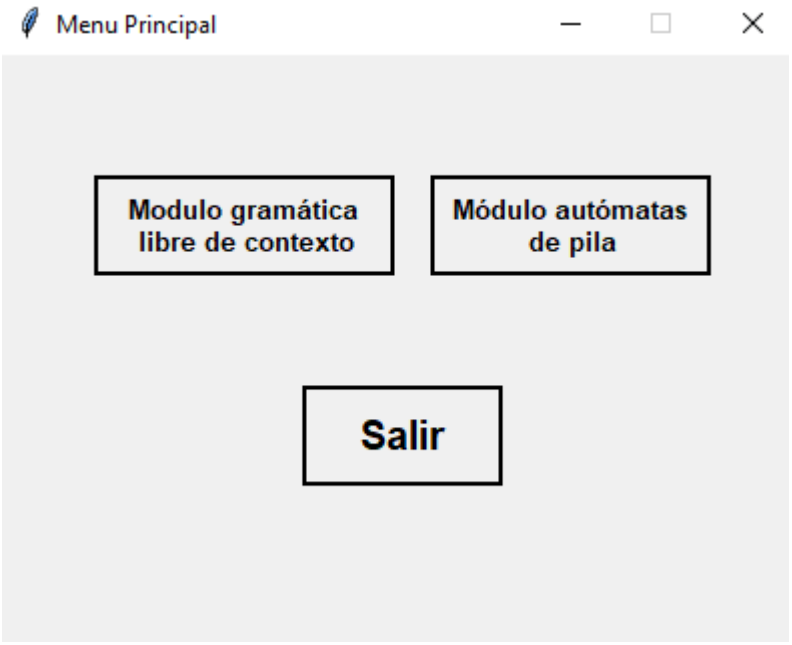
4.2 Tkinter

Se uso la librería Tkinter para la interfaz gráfica del proyecto, se decidió usar debido a que esta librería se pide como requisito para la elaboración del proyecto y próximos trabajos a realizar en el curso.

```
menu = Tk()
menu.title('Proyecto 1 - Lenguajes Formales A+')
menu.geometry("1700x1000")

opciones = Menubutton(menu,text='Opciones Archivo',width=15,height=3,font=('Arial',15),cursor='hand2',bg='#088BD1',relief=SOLID,activebackground='#2CAFF5')
opciones.place(relx=0.01,rely=0.02)
opcion =Menu(opciones,tearoff=0,font=('Arial',13),cursor='hand2',bd=15,bg='#2CAFF5',activebackground='#F78411',activeforeground='#000000')
opcion.add_command(label='Guardar',command=guardar)
opcion.add_separator()
opcion.add_command(label='Guardar Como',command=guardar_como)
opcion.add_separator()
opcion.add_command(label='Errores',command=Rep_Errores)
opcion.add_separator()
opcion.add_command(label='Salir',command=exit)
opcion.add_separator()
opciones.config(menu=opcion)

ayuda = Menubutton(menu,text='Ayuda',width=12,height=3,font=('Arial',15),cursor='hand2',bg='#23BC0E',relief=SOLID,activebackground='#50DE3D')
ayuda.place(relx=0.13,rely=0.02)
opcion2 =Menu(ayuda,tearoff=0,font=('Arial',13),cursor='hand2',bd=15,bg='#50DE3D',activebackground='#F78411',activeforeground='#000000')
opcion2.add_command(label='Manual de Usuario',command=a)
opcion2.add_separator()
opcion2.add_command(label='Manual Técnico',command=a)
opcion2.add_separator()
opcion2.add_command(label='Temas de Ayuda',command=a)
opcion2.add_separator()
ayuda.config(menu=opcion2)

frame1 = Frame(menu)
frame1.place(relx=0.01,rely=0.12)
cuadro_archivo = Text(frame1,width=100,height=40,font='Arial')
cuadro_archivo.grid(row=0,column=0)
SC_archivo = Scrollbar(frame1, command=cuadro_archivo.yview)
cuadro_archivo['yscroll'] = SC_archivo.set
SC_archivo.grid(row=0,column=2)

frame2 = Frame(menu)
frame2.place(relx=0.58,rely=0.12)
cuadro_texto = Text(frame2,width=70,height=20,font='Arial')
cuadro_texto.grid(row=0,column=0)
SC_texto = Scrollbar(frame2, command=cuadro_texto.yview)
cuadro_texto['yscroll'] = SC_texto.set
SC_texto.grid(row=0,column=2)
```

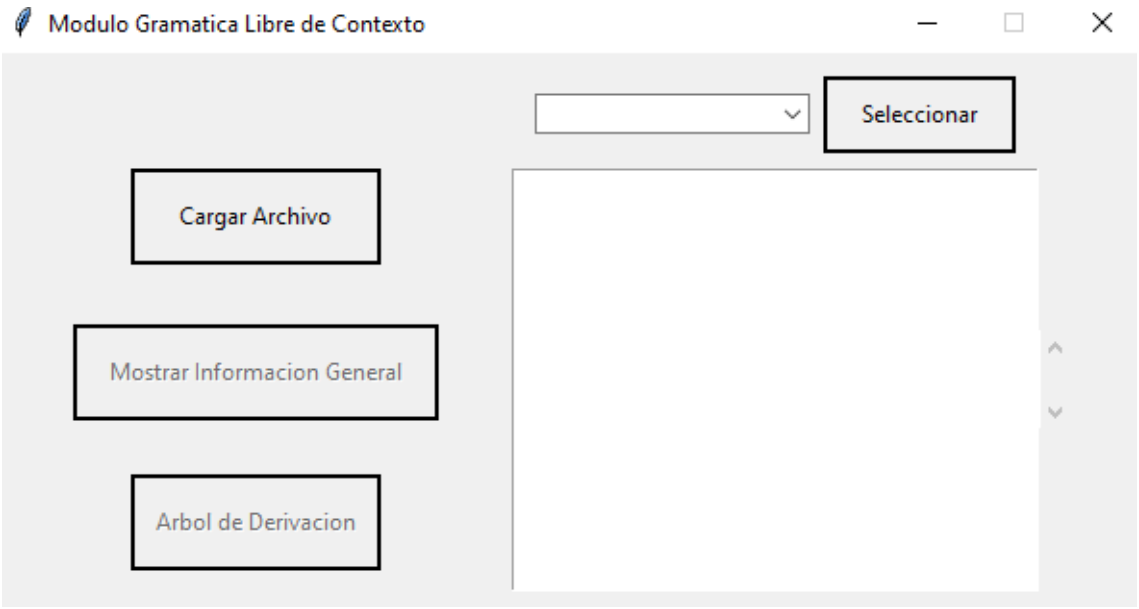
5. Interfaces Principales

5.1 Menú Principal



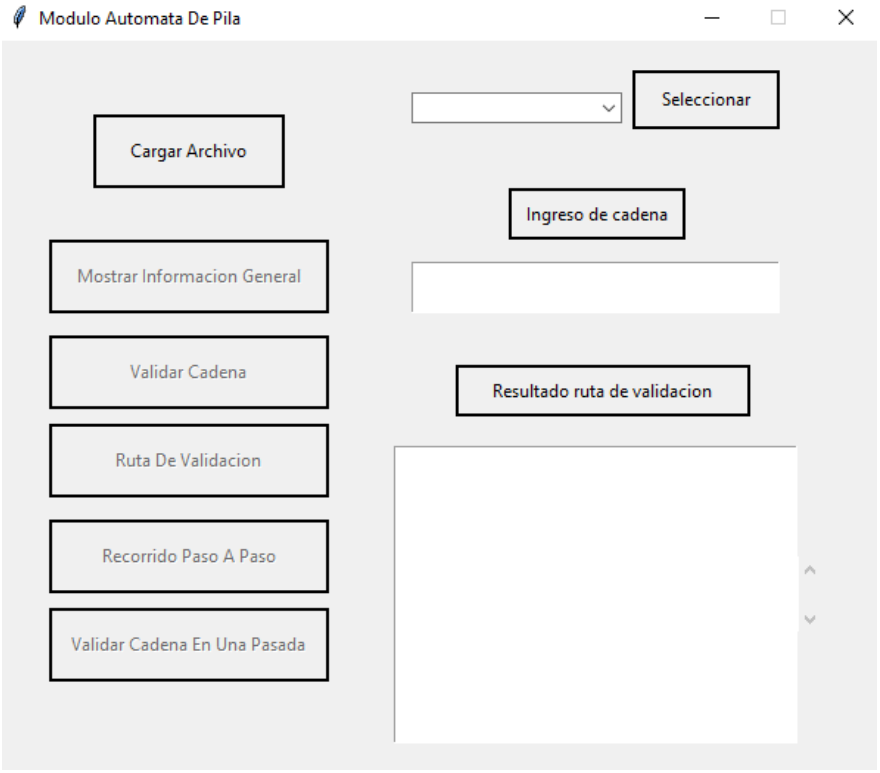
En esta primera interfaz se muestran las opciones que se tendrán en el programa.

5.2 Modulo Gramáticas Libre de Contexto



En esta interfaz se muestran las acciones que se pueden hacer después de haber cargado el archivo de entrada.

5.3 Modulo Autómata de Pila



En esta interfaz se muestran las acciones que se pueden realizar después de haber cargado el archivo de entrada.

6. Planificación o Estimación

6.1 Tarea #1
[Creación de interfaz)] ---------- [2 Horas]

6.2 Tarea #2
[Creación de métodos y funciones] ---------- [4 Horas]

6.3 Tarea # 3
[Validaciones] ---------- [2 Horas]

7. Glosario

I. IndexError: es un error usual al momento de agregar un valor de índice fuera del rango de nuestras listas.
II. Identación: este término significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores, para así separarlo del margen izquierdo y distinguirlo mejor del texto adyacente
III. ValueError: es un error que sucede cuando trato de convertir un valor a otro tipo que no es posible, por ejemplo, una cadena de caracteres a números.
IV. Importaciones de orden Superior: un error que sucede cuando se trata de importar algo que no está al mismo nivel que la raíz.