UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

LENGUAJES FORMALES Y DE PROGRMACIÓN

MANUAL TÉCNICO

PROYECTO 2: AUTOMATA DE PILA

ELABORADO POR: EDMY MARLENY MENDOZA POL

ABRIL 2021

INTRODUCCION

El presente manual se ha fabricado con el propósito de mostrar cómo fue diseñado el programa, cual es el proceso para generar un PDA a partir de una gramática libre de contexto.

La aplicación desarrollada permite la lectura de un archivo glc, el cual contendrá *n* número de gramáticas, para las cuales se podrán realizar las siguientes operaciones principales:

1. Leer Gramática
2. Generar PDA
3. Generar Reportes: Por recorrido o en una Tabla.

Requerimientos de Software:

Python: versión 3.8.5

Privilegios de administrador: SI

Sistema operativo: Windows 7, 8, 10, Mac OS, Linux

Navegador Web

Requerimientos de Hardware:

Memoria RAM: 2Gb

Disco Duro: 50mb libre

Herramientas Utilizadas para el desarrollo:

Python: versión 3.8.5

IDE: Visual Basic

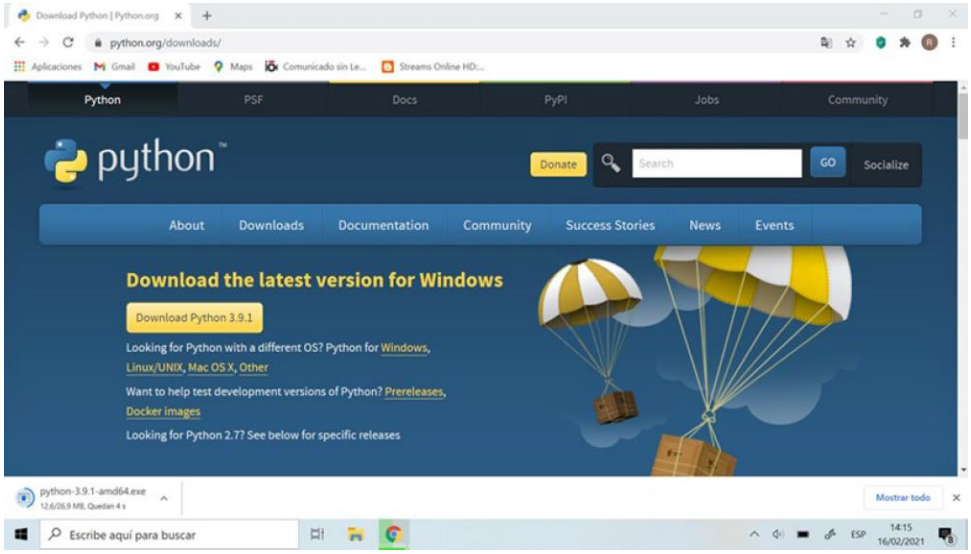
Navegador Web: Google Chrome

Librerías Usadas:

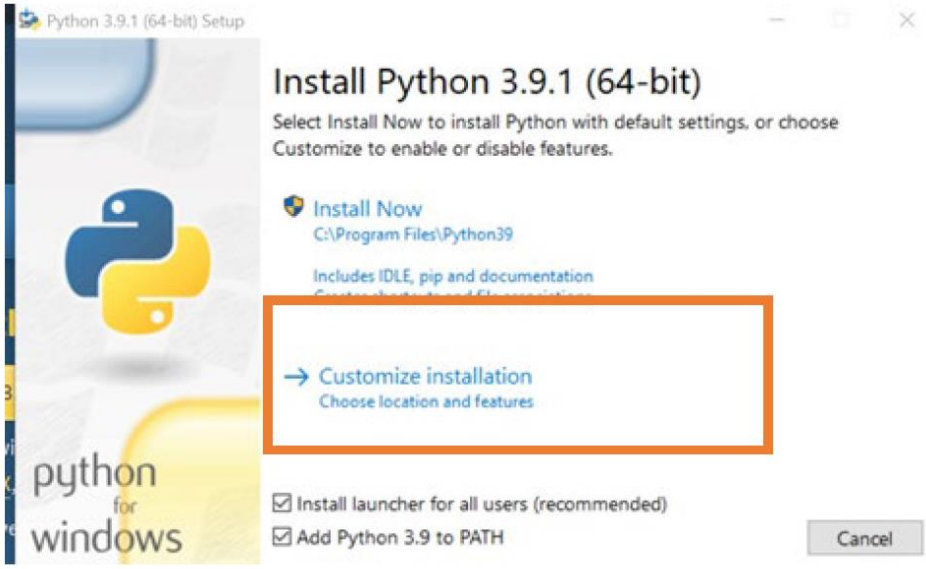
- Tkinter
- Webbrowser
- Graphviz
- Os

Instalación:

Descargar Python según el sistema operativo poseído del siguiente enlace: https://www.python.org/downloads/release/python-385/

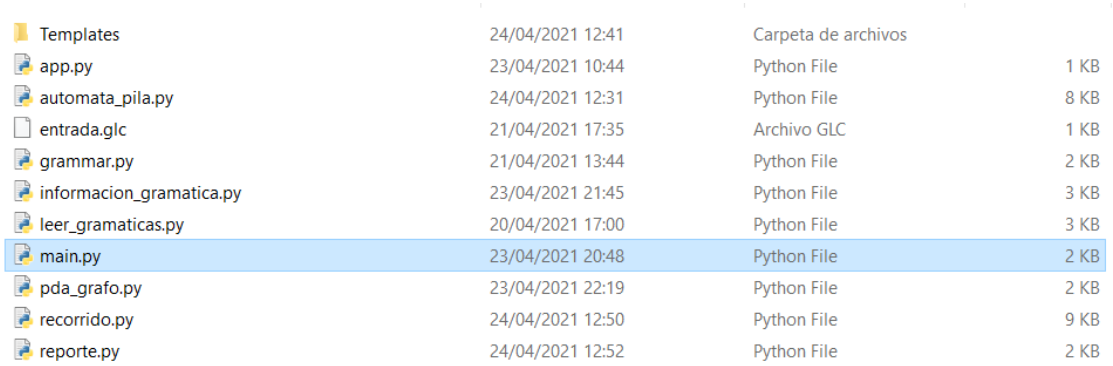


Ejecutar Python.exe

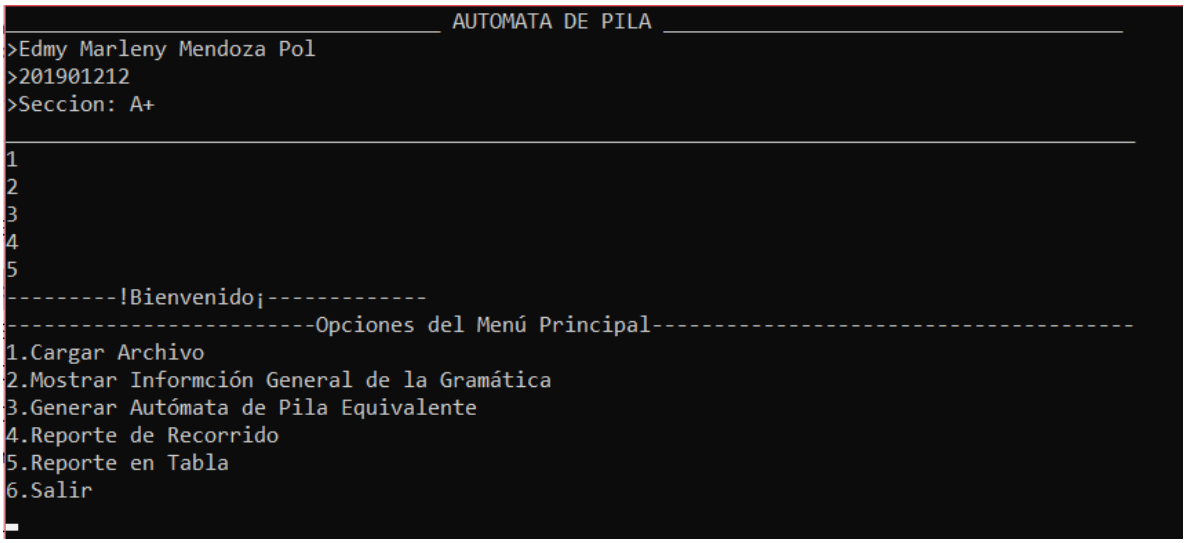


Deberá seguir las instrucciones del instalador, en cuestión de minutos Python estará instalado en su sistema operativo.

Descomprimir el archivo [LFP]Proyecto2_201901212 y ejecutar main.py

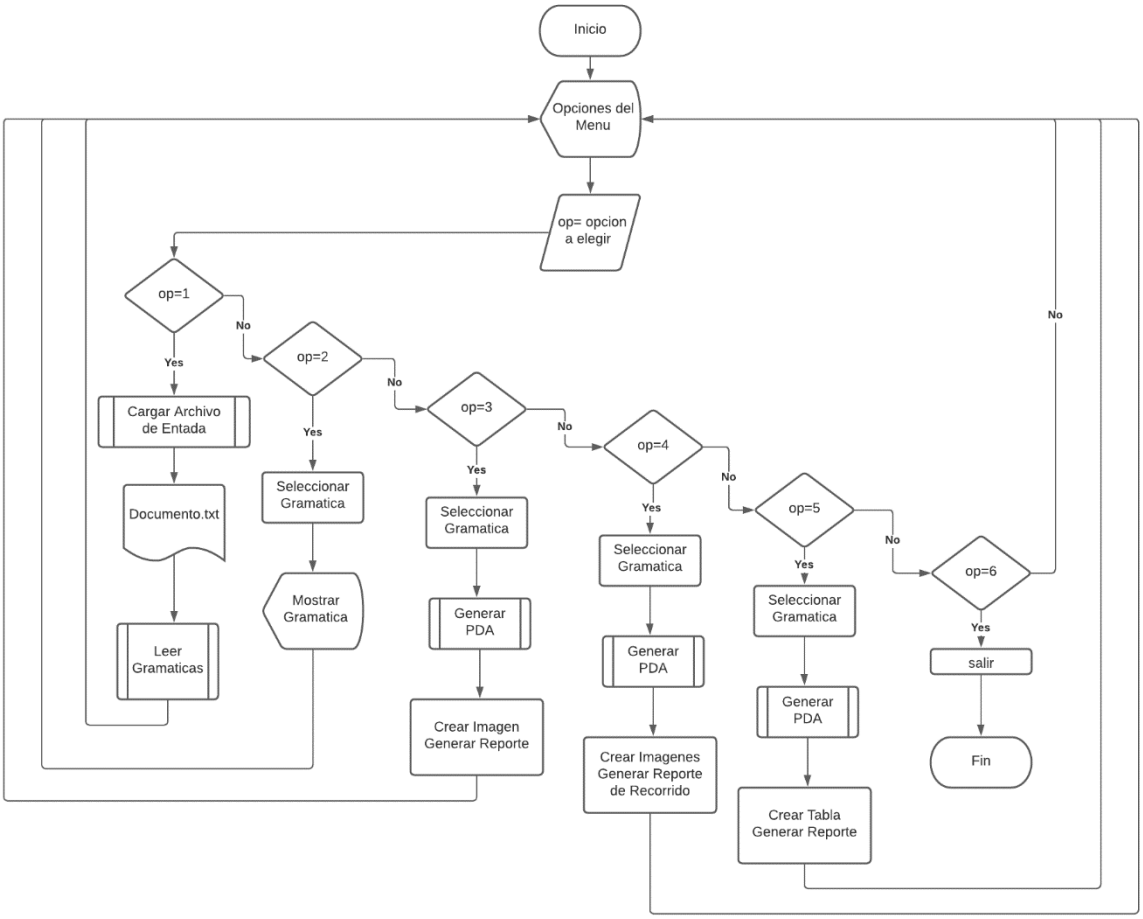


Aparecerá el menú principal después de esperar 5 segundos.



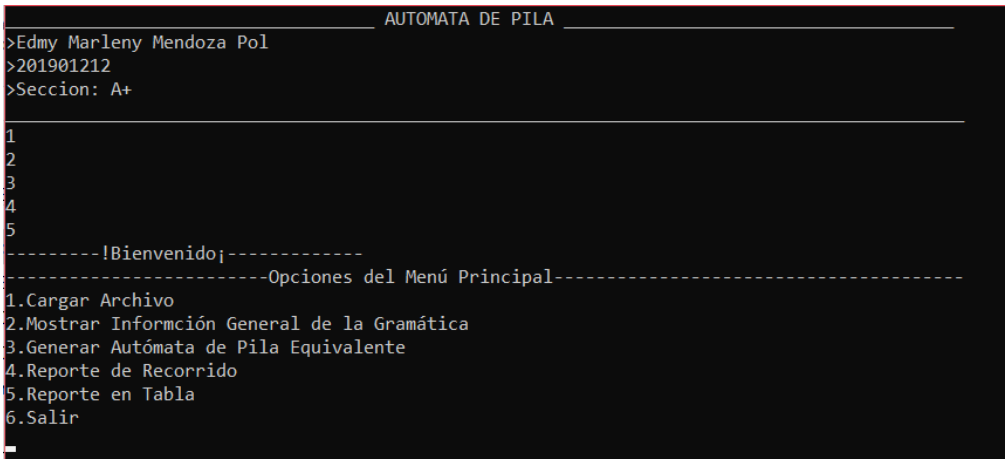
Procesos y Métodos:

Flujo General de la Aplicación:



Menú Principal

Después de ejecutar la aplicación se le mostraran al usurario los datos estudiantiles del programador, tras lo cual deberá esperar 5 segundos para poder visualizar las 6 opciones disponibles, para elegir una de ellas el usuario debe presionar el numero correspondiente por medio del teclado



Funcionamiento del Menú:

Se importan las funciones que corresponden a cada acción que puede realizar el usuario:

```
import time
from leer_gramaticas import leer_gramaticas
from automata_pila import pda
from automata_pila import iterar
from informacion_gramatica import mostrar_gramatica
from informacion_gramatica import seleccionar_gramatica
from pda_grafo import grafo_pda
from reporte import reportar
from recorrido import grafo_pda_recorrido
```

Desde aquí se llevará el control de cuáles son las acciones que el usuario desea realizar.

La función esperar imprime un numero cada segundo hasta llegar al número cinco. Para hacer el menú se creó un ciclo while, usando como condicional la variable global “seguir”, el programa se ejecutará mientras esta variable sea verdadera.

```
def esperar():
    for n in range(1,6):
        time.sleep(1)
        print(n)

while seguir:
    print("_____ AUTOMATA DE PILA _____")
    print(">Edmy Marleny Mendoza Pol\n>201901212\n>Seccion: A+")
    print("_____")
    esperar()
    print("---------¡Bienvenido¡----------")
    print("------------------------Opciones del Menú Principal-----------------------------------")
    print("1.Cargar Archivo\n2.Mostrar Informción General de la Gramática\n3.Generar Autómata de Pila Equivalente")
    print("4.Reporte de Recorrido\n5.Reporte en Tabla\n6.Salir")

    op = input()

    if op=='1':
        gramaticas = leer_gramaticas()
    elif op=='2':
        mostrar_gramatica(gramaticas)
    elif op=='3':
        grafo_pda(gramaticas)
    elif op=='4':
        gramatica = seleccionar_gramatica(gramaticas)
        iteraciones = pda(gramatica)
        grafo_pda_recorrido(gramatica,iteraciones)
        print()
    elif op=='5':
        gramatica = seleccionar_gramatica(gramaticas)
        ap = pda(gramatica)
        reportar("",ap)
    elif op=='6':
        seguir=False
```

Como se mencionó en la introducción, a rasgos generales la aplicación tiene 3 funciones principales:

1. Leer Gramática:

Se comienza por preguntar a usuario por la ruta del archivo que contiene las gramáticas.

1.1 Cargar Archivo:

```
from tkinter.filedialog import askopenfilename
from tkinter import Tk
```

```
def cargar():
    Tk().withdraw()
    ruta = askopenfilename()
    return ruta
```

Esta función crea una interfaz gráfica por medio del paquete tkinter que permitirá al usuario elegir un archivo de entrada a través del explorador de archivos. Devuelve la ruta del archivo seleccionado por el usuario.

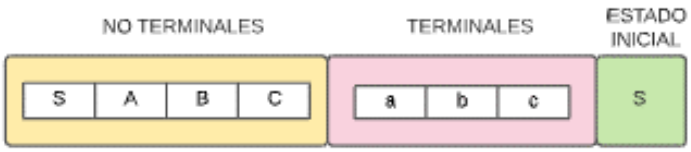
1.2 Leer Gramáticas:

```
def leer_gramaticas():
    root = cargar()
    f = open (root,'r')
    n=0
    gramaticas = []

    for linea in f:
      if linea[0]!=" ":
        if n==0:
            name = linea.strip("\n")
            producciones = []
            n+=1
        elif n==1:
            terminos = linea.split(";")
            no_terminales= terminos[0].split(",")
            terminales = terminos[1].split(",")
            terminal_inicial = terminos[2].strip("\n")
            n+=1
        elif n==2 and linea[0]!="*":
            produc = linea.split("->")
            der_produc = produc[1].split()
            producs = definir_termino(terminales,no_terminales,der_produc)
            prod = produccion(produc[0],producs)
            producciones.append(prod)

        elif linea[0]=="*":
            n=0
            reglas = agrupar_producciones(no_terminales,producciones)
            grammar = gramatica(name,reglas,terminales,no_terminales,terminal_inicial)
            add_grammar = validar_gramatica(grammar)
            if add_grammar == 'SI':
                gramaticas.append(grammar)


    f.close()

    return gramaticas
```

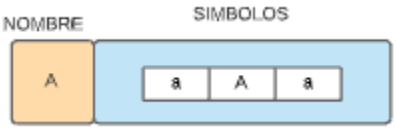
Esta función leerá línea por línea el archivo cargado y devuelve una lista que contiene todas las gramáticas libres de contexto del archivo.

```
Grm1
S,A,B,C;a,b,z;S
S->A
A->a A a
A->B
B->b B b
B->C
C-> z C
C->z
*
```

- En la primera línea se encontrará el nombre de la gramática el cual lo almacenará en la variable name. También se inicializará una lista que contendrá todas las producciones de la gramática.
- En la segunda línea primero se realizará una separación cada “; “, después esas separaciones se vuelven a separar cada “,”. De esta manera se obtendrá una lista con los términos no terminales, otra lista con los símbolos no terminales y el estado inicial.



- Cada línea a partir de la tercera se separa en 2 con el símbolo “->”, la primera parte contiene el nombre de la producción, la segunda parte contiene cada uno de los símbolos que forman la producción separados por un espacio.



Con ayuda de la función definir termino, crearemos un objeto símbolo por cada elemento de la producción, este objeto llevara como parámetros el carácter que lo conforma y si es terminal o no.

```
class simbolo:
    def __init__(self, sim = None, terminal = None):
        self.sim = sim
        self.terminal = terminal
```

```
class produccion:
    def __init__(self,produc_nombre= None, rules = None):
        self.produc_nombre=produc_nombre
        self.rules = rules
```

```
def definir_termino(terminales,no_terminales,der_produc):
    definido = []

    for term in der_produc:
        for ter in terminales:
            if ter == term:
                simb = simbolo(term,"SI")
        for termino in no_terminales:
            if termino == term:
                simb = simbolo(term,"NO")

        definido.append(simb)

    return definido
```

La función definir termino devuelve una lista con los símbolos que conforman la producción. Luego que crea un objeto producción usando de parámetros el nombre y la lista de símbolos. Luego se agrega esta producción a la lista de producciones

- En el momento en el que se encuentre el símbolo “*” se dará por terminada la gramática. Se agrupan las producciones con el mismo nombre por lo que un solo termino terminal puede tener más de una derivación.

```
def agrupar_producciones(no_terminales, producciones):

    productions = []

    for nt in no_terminales:
        production = []
        for p in producciones:
            if p.produc_nombre == nt:
                production.append(p.rules)

        pros = produccion(nt,production)
        productions.append(pros)

    return productions
```

Luego se crea la gramática con los datos: nombre, producciones, símbolos terminales y no terminales, así como su estado inicial.

```
class gramatica:
    def __init__(self,nombre = None, producciones = None, sim_terminales = None, sim_Noterminales= None, state_inicial=None):
        self.nombre = nombre
        self.producciones = producciones
        self.sim_terminales = sim_terminales
        self.sim_Noterminales = sim_Noterminales
        self.state_inicial = state_inicial
```

Una vez creada esta gramática se valida que esta sea libre de contexto y no regular. Esto se hace verificando que al menos una de sus producciones tenga un símbolo terminal al inicio del lado derecho. Si es libre de contexto se añade a la lista de gramáticas

```
def validar_gramatica(grammar):
    libre_contexto = "NO"

    for regla in grammar.producciones:

        for rule in regla.rules:
            size = len(rule)
            for r in range(size):
                print(r)
                if r!=1:
                    if rule[r].terminal == 'NO':
                        libre_contexto = 'SI'
    return libre_contexto
```

Una vez se tiene el listado de gramáticas podemos realizar la opción de:

Mostrar Información General de la Gramática

Primero debemos de **seleccionar** cual es la gramática que deseamos mostrar:

```
def seleccionar_gramatica(gramaticas):
    borrarPantalla()
    print("Estas son las Gramáticas Cargadas en Memoria:")
    n = 0
    for gra in gramaticas:
        n+=1
        print(str(n)+".)"+gra.nombre)

    print()
    print("Ingrese el numero correspondiente a la gramatica a seleccionar:")
    nombre = input()
    n=0

    for g in gramaticas:
        n+=1
        if str(n) == nombre:
            gramatica = g

    return gramatica
```

Esta función imprime los nombres las gramáticas en memoria y devuelve la gramática elegida por el usuario.

```
def mostrar_gramatica(gramaticas):
    gramatica = seleccionar_gramatica(gramaticas)
    print()
    print("_____")
    print("Nombre de la Gramatica tipo 2: "+gramatica.nombre)
    print("No terminales: ",end="")
    print(gramatica.sim_Noterminales)
    print("Terminales: ",end="")
    print(gramatica.sim_terminales)
    print("No terminal Inicial: ",gramatica.state_inicial)
    print("Producciones: ")
    gramatica.imprimir_gramatica()
```

La función mostrar gramática recibe por parámetro la gramática seleccionada e imprime en consola: el nombre de la gramática, símbolos terminales y no terminales, el símbolo inicial y las producciones correspondientes.

2. Generar Autómata de Pila

Primero se selecciona la gramática con la función expuesta anteriormente. La función PDA recibe de parámetro la gramática seleccionada. El PDA cuenta con 4 estados: [i, p, q, f]

```
def pda(gramatica):


    print("Introduzca la palabra a evaluar")
    entrada = input()
    length = len(entrada)
    state = "i"
    pila = []
    error = ""
    i=0
    k=0
    iteraciones =  []
```

En la primera parte de la función, se solicita la cadena a evaluar, se obtiene el tamaño de esta. Se inicializa el PDA en el estado “i”, también se inicializa una lista que cumplirá la función de una pila. La variable “i” sirve para mover el apuntador de la cadena, la variable k sirve para enumerar las iteraciones que realiza el PDA.

```
class iteracion:
    def __init__(self,id= None, pila = None, entrada = None, transicion= None, cambio=None):
        self.id = id
        self.pila = pila
        self.entrada = entrada
        self.transicion = transicion
        self.cambio = cambio
```

Cada vez que el PDA use una producción, se agregara a una lista una iteración cuyo contenido es: el número de iteración, lo que hay en la pila, el carácter de la entrada que se está evaluando, que producción fue aplicada.

Luego se inicia un ciclo while que seguirá hasta que se haya recorrido por completo la cadena o haya ocurrido un error.

Estado i:

```
while (i<=length):

    if state=="i":
        valido = simbolo("#","NO")
        pila.append(valido)
        state = "p"

        pile = obtener_pila(pila,i,entrada)

        it = iteracion('0'," ",entrada[0],"(i,$,$;p,#)","$ , $ ; #")
        iteraciones.append(it)
```

Este es el estado inicial, aquí se inicializará la pila con el símbolo de aceptación “#”. También se agregará una iteración. La función obtener pila devuelve una cadena conformada por los elementos de la pila en el orden inverso para ser utilizado en los reportes.

```
def obtener_pila(pila,i,entrada):
    pile = ""

    pila.reverse()
    for elemento in pila:

        pile+=elemento.sim
    pila.reverse()


    return pile
```

De este estado se pasa al estado “p”.

Estado p:

```
elif state =="p":

    inicial = simbolo(gramatica.state_inicial,"NO")
    pila.append(inicial)
    state = "q"
    k+=1
    it = iteracion(str(k),"#",entrada[i],"(p,$,$;q,"+gramatica.state_inicial+")","$ , $ ; "+gramatica.state_inicial)
    iteraciones.append(it)
```

En este estado se agrega a la pila el símbolo inicial, también se agrega otra iteración y se pasa al estado “q”.

Estado q:

En este estado es donde se realizan las derivaciones. Aquí pueden suceder varios casos el primero de ellos es que el símbolo en la cima de la pila sea no terminal.

A. Cima de la pila: No Terminal:

Si esto sucede, primero se verifica que cuantas posibles derivaciones tiene este símbolo.

Una sola derivación:

```
elif state == "q":
    for produccion in gramatica.producciones:
        if pila[-1].terminal=="NO" and pila[-1].sim==produccion.produc_nombre:

            rel = ""
            inv = ""
            if len(produccion.rules)==1:
                pile = obtener_pila(pila,i,entrada)
                pila.pop()
                for elem in produccion.rules:
                    elem.reverse()

                    for r in elem:
                        pila.append(r)
                        rel+=r.sim
                        inv = r.sim + inv
                    elem.reverse()

                    k+=1
                    it = iteracion(str(k),pile,entrada[i],"(q,$,"+produccion.produc_nombre+";q,"+rel+")","$ , "+produccion.produc_nombre+" ; "+inv)
                    iteraciones.append(it)
```

Si se da el caso de que al símbolo no terminal de la pila le corresponde una sola derivación, entonces se ingresara cada elemento de la producción en el orden inverso y se agregara una iteración.

Dos o más Derivaciones:

```
else:
    regla = None
    cambiado = False
    op = False

    for rule in produccion.rules:

        if rule[0].sim == entrada[i]:
            cambiado = True
            if len(rule)!=1 and i+1<=length:
                if rule[1].sim == entrada[i+1]:
                    regla = rule
                    op = True
                elif regla == None:
                    regla = rule
            elif regla == None or op == False:
                regla = rule


    if cambiado == False:
            opti = False
            for rule in produccion.rules:
                if rule[0].terminal == "NO":
                    for prod in gramatica.producciones:
                        if rule[0].sim == prod.produc_nombre:
                            for produc in prod.rules:
                                if produc[0].sim == entrada[i]:

                                    if len(produc)!=1 and i+1<=length:

                                        if produc[1].sim == entrada[i+1]:

                                            regla = rule
                                            opti = True
                                        elif regla == None:
                                            regla = rule
                                    elif regla == None or opti == False:
                                            regla = rule

                    if regla == None or opti == False:
                        regla = rule
```

La variable “regla” contendrá la producción a usar.

De darse el caso de un símbolo no terminal con más de una derivación, primero se busca si una de ellas tiene como primer símbolo un carácter igual al de la cadena de entrada en la posición “i”. Luego se analiza si el segundo símbolo de esa producción es igual al carácter de la entrada en la posición “i+1”, de darse el caso gracias a la variable “op” esta será la producción para usar. De lo contrario se usará la derivación en la que al menos un símbolo coincide con la entrada, la variable “cambiado” cambiara a verdadera, lo que nos indicara que se ha encontrado una regla.

Si la variable cambio sigue siendo falsa, esto indica que no se ha encontrado una derivación cuyo primer símbolo encaje con la entrada, por lo que a continuación se buscara si una de las derivaciones tiene como primer símbolo uno No terminal de ser este el caso, se busca entre las derivaciones de este nuevo símbolo si alguna una de ellas tiene como primer símbolo un carácter igual al de la cadena de entrada en la posición “i”, es decir se repite el proceso descrito anteriormente.

```
if regla!=None:
    regla.reverse()
    pile = obtener_pila(pila,i,entrada)
    pila.pop()
    rel =""
    inv = ""
    for el in regla:
        pila.append(el)
        rel+=el.sim
        inv = el.sim + inv


    regla.reverse()
    k+=1
    it = iteracion(str(k),pile,entrada[i],"(q,$,"+pila[-1].sim+";q,"+rel+")","$ , "+produccion.produc_nombre+" ; "+inv)
    iteraciones.append(it)
```

Una vez completado el proceso anterior, se habrá obtenido una derivación. Por lo que se procede a ingresar los símbolos de esta a la pila en el orden inverso. También se agrega una iteración.

B. Cima de la Pila: Símbolo Terminal igual al carácter en la posición evaluada:

```
elif pila[-1].terminal=="SI" and pila[-1].sim == entrada[i]:

    k+=1
    pile = obtener_pila(pila,i,entrada)
    it = iteracion(str(k),pile,entrada[i],"(q,"+entrada[i]+","+entrada[i]+";q,$)",entrada[i]+" , "+entrada[i]+"; $")
    iteraciones.append(it)
    pila.pop()
    i+=1
```

De ser este el caso, entonces se elimina el símbolo terminal de la pila y se avanza un carácter en la cadena a evaluar. Se agrega una iteración.

C. Cima de la Pila: Símbolo Terminal Diferente al carácter en la posición evaluada:

```
elif pila[-1].terminal=="SI" and pila[-1].sim != entrada[i]:
    error = "ERROR = El Simbolo Terminal en la Cima de la pila No coincide con el Caracter a evaluar en la posición: "+str(i)+" de la cadena"

    state = "f"
    break
```

Si se presenta este caso se estaría tratando de un error, por lo cual guardamos el error y pasamos al estado “f” donde NO se aceptará la cadena, simplemente se usará para guardar la iteración.

D. En la pila solo queda el símbolo de aceptación:

```
elif len(pila)==1 and pila[0].sim=="#":
    pila.pop()
    k+=1

    if len(entrada)>i:
        entry = entrada[i]
    else:
        entry = "$"
    it = iteracion(str(k),"#",entry,"(q,$,#;f,$)","-----")
    iteraciones.append(it)
    state = "f"
    break
```

De ser este el caso, se elimina el símbolo de aceptación de la pila por lo cual queda vacía. Se agrega una iteración y se pasa al estado “f”.

Estado f:

```
elif state =="f":

    if i==length and error=="":
        k+=1
        it = iteracion(str(k)," $ "," $ ","f","-----")
        iteraciones.append(it)
        it = iteracion("#","AP_"+gramatica.nombre,"","La Cadena = "+entrada+"  fue Aceptada",entrada)
        iteraciones.append(it)

    elif error !="" and i!=length:
        it = iteracion("&","AP_"+gramatica.nombre,error,"La Cadena = "+entrada+" No fue Aceptada",entrada)
        iteraciones.append(it)

    else:
        it = iteracion("&","AP_"+gramatica.nombre,"ERROR = En la cima de la pila esta el simbolo de aceptación,  pero no se ha terminado de recorrer la cadena ","La Cadena = "+en
        iteraciones.append(it)

    i=length+2

return iteraciones
```

En este esta se aceptará la cadena como valida solo si la variable error está vacía y el contador “i” es igual. De lo contrario esta cadena no es aceptada, por lo que se toma como un error.

3. Generar Reportes:

3.1 Generar autómata de pila equivalente a través de un grafo.

En esta opción el usuario deberá elegir una gramática cargada en el sistema y a continuación generar un reporte en HTML que muestre el autómata de pila equivalente a través de un grafo.

Ya se explicó con anterioridad cómo funciona el pda. Así que aquí se expondrá como se realiza el reporte del autómata.

```
def grafo_pda(gramaticas):
    gramatica = seleccionar("","las Gramaticas ",gramaticas)

    dott = """digraph {
        graph [rankdir=LR]
        i [label=i shape=circle style=filled fillcolor=skyblue]
        p [label=p shape=circle style=filled fillcolor=skyblue]
        q [label=q height=1 shape=circle width = 1 style=filled fillcolor=skyblue]
        f [label=f shape=doublecircle style=filled fillcolor=skyblue]"""

    etq = label_grafo(gramatica)
    dott+=etq

    etq1= llenar_transformaciones(gramatica)
    etq2 = llenar_ter(gramatica.sim_terminales)

    dott+='i -> p [label ="$ , $ ; #"]'
    dott+='p -> q [label="$ , $ ; '+gramatica.state_inicial+'"]'
    dott+='q:s -> q:s [label="'+etq2+'"]'
    dott+='q:n -> q:n [label="'+etq1+'"]'
    dott+= 'q -> f [label="$ , # , $"]}'

    nom = 'AP_'+gramatica.nombre



    src = Source(dott)
    src.render("Templates/"+nom, format = 'png')

    generar_pda(nom)
```

Se usa el lenguaje DOT para formar un grafo que cuente con los datos de la gramática utilizada, así como sus producciones.

```
def llenar_transformaciones(gramatica):

    transformaciones = ""
    for produccion in gramatica.producciones:

        for rule in produccion.rules:
            transformaciones+="$ , "+produccion.produc_nombre+" ; "
            for r in rule:
                transformaciones+=r.sim
            transformaciones+="\n"

    return transformaciones

def llenar_ter(terminales):
    ter = ""
    for t in terminales:
        ter+=t+" , "+t+"; $\n"

    return ter
```

Las funciones "*llenar_transformaciones"* y *"llenar_ter"* devuelven una cadena con las producciones de la gramática en un formato adecuado para su visualización.

3.2 Reporte de Recorrido:

El usuario podrá elegir uno de los autómatas de pila generados. Finalizada la elección del autómata, se solicitará al usuario el ingreso de una cadena de entrada para que sea validada con el autómata. Esta opción permitirá generar un reporte en HTML que contendrá a detalle cada una de las iteraciones realizadas para validar la cadena.

Para crear el recorrido, utilizando el lenguaje DOT y graphviz se creó una imagen png por cada iteración realizada por el autómata. Luego estas imágenes se colocaron en una tabla HTML para darles formato.

```
def grafo_pda_recorrido(gramatica,iteraciones):


    for iteracion in iteraciones:

        if iteracion.transicion == "f":
            dott = definir_estado("f",gramatica.state_inicial,iteracion,len(iteraciones))
        else:
            dott = definir_estado(iteracion.id,gramatica.state_inicial,iteracion,len(iteraciones))

        ter = colorear_terminos(gramatica.sim_terminales,iteracion)
        noter = colorear_noterminales(gramatica, iteracion)


        dott+="q:s -> q:s [label=<<table BORDER='0' CELLBORDER='0' CELLSPACING='0'>"+ter+"</table>>]"
        dott+="q:n -> q:n [label=<<table BORDER='0' CELLBORDER='0' CELLSPACING='0'>"+noter+"</table>>]}"


        if iteracion.id != "#" and iteracion.id!="&":
            nom = "Templates/imagen"+iteracion.id
            src = Source(dott)
            src.render(nom, format = 'png')

    reportar_recorrido(gramatica.nombre,iteraciones)
```

La función *definir estado* se encarga de colorear el estado actual según la iteración también se define aquí la tabla que mostrara los símbolos en la pila en esa iteración, así como la entrada en la posición a evaluar.

```
def definir_estado(e,inicial,iteracion,m):
    estados = ""

    if e == "0":
        estados =  """digraph {
        graph [rankdir=LR]
        i [label=<i> shape=circle style=filled fillcolor=skyblue]
        p [label=p shape=circle style=filled fillcolor=azure2]
        q [label=q height=1 shape=circle width = 1 style=filled fillcolor=azure2]
        f [label=f shape=doublecircle style=filled fillcolor=azure2]
        i -> p [label =<<FONT COLOR="blue">$ , $ ; #</FONT>>]
        q -> f [label =<$ , # ; $>]
        p -> q [label =<$ , $ ; """
        estados+=inicial+">]"

        estados+= """a [shape = none label=<<table border="0" cellborder="1" cellspacing="0"
<tr><td BGCOLOR="lightblue1">Iteracion</td><td BGCOLOR="lightblue1"><b>0</b></td></tr>
<tr><td BGCOLOR="lightblue1" >Pila</td><td>  </td></tr>
<tr><td BGCOLOR="lightblue1" >Entrada</td><td>  </td></tr>
</table>>]"""
    elif e== "1":
        estados =  """digraph {
        graph [rankdir=LR]
        i [label=<i> shape=circle style=filled fillcolor=azure2]
        p [label=p shape=circle  style=filled fillcolor=skyblue]
        q [label=q height=1 shape=circle width = 1 style=filled fillcolor=azure2]
        f [label=f shape=doublecircle style=filled fillcolor=azure2]
        i -> p [label =<$ , $ ; #>]
        q -> f [label =<$ , # ; $>]"""
        estados +='p -> q [label =<<FONT COLOR="blue">$ , $ ; '+inicial+'</FONT>>]'
```

Las funciones *colorear_terminos* y *colorear_noterminales* se encargan de colorear la regla utilizada en esa iteración.

```
def colorear_noterminales(gramatica,iteracion):

    noter = ""
    etq1 = []
    for produccion in gramatica.producciones:

        for rule in produccion.rules:
            transformaciones="$ , "+produccion.produc_nombre+" ; "
            for r in rule:
                transformaciones+=r.sim

            etq1.append(transformaciones)

    for a in etq1:
        if a == iteracion.cambio:

            noter+='<tr><td><FONT COLOR="blue">'+a+'</FONT></td></tr>'
        else:
            noter+='<tr><td>'+a+'</td></tr>'

    return noter


#-------------------------------------------------------------------------
def colorear_terminos(terminales, iteracion):
    ter = ""
    etq2 = []
    for t in terminales:
        termi=t+" , "+t+"; $"
        etq2.append(termi)

    for a in etq2:
        if a == iteracion.cambio:

            ter+='<tr><td><FONT COLOR="blue">'+a+'</FONT></td></tr>'
        else:
            ter+='<tr><td>'+a+'</td></tr>'

    return ter
```

3.3 Reporte en Tabla:

El usuario podrá elegir uno de los autómatas de pila generados en la opción 3. Finalizada la elección del autómata, se solicitará al usuario el ingreso de una cadena de entrada para validar si se acepta no en el autómata. Esta opción deberá mostrar como resultado un reporte en HTML con una tabla de resumen.

La función reportar, escribe un documento HTML con una tabla que contiene cada una de las iteraciones realizadas por el autómata.

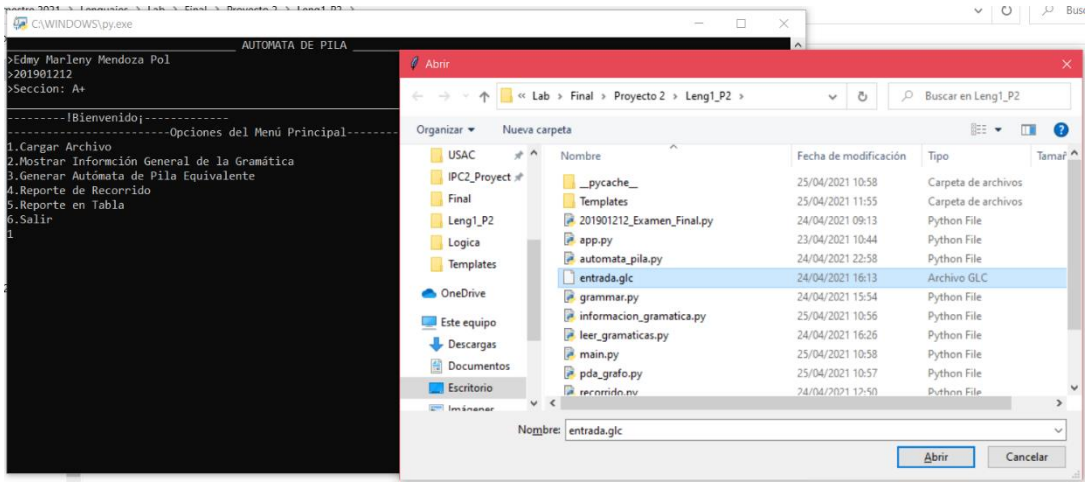
```
def reportar(nombre, iteraciones):
  contenido = escribir_reporte(nombre,iteraciones)
  t = open("Templates/template_reporte.html",'r')
  f = open('Reporte_Tabla_'+iteraciones[-1].pila+'.html','w',encoding="utf-8")
  template = t.read()

  cuerpo = template % (contenido)
  f.write(cuerpo)
  t.close()
  f.close()

  webbrowser.open_new_tab('Reporte_Tabla_'+iteraciones[-1].pila+'.html')
```

Para realizar la tabla se utiliza la función escribir reporte, la cual se encargará de llenar la tabla HTML con la información de: el número de iteración, los elementos de la pila, la entrada, transiciones. Al finalizar la tabla se escribirá si esta la cadena fue aceptada o no.

```
def escribir_reporte(nombre, iteraciones):
    encabezados = "<h1>Tabla: "+iteraciones[-1].pila+"</h1>"
    encabezados += "<h2>"+iteraciones[-1].transicion+"</h2>"

    encabezados += """<center>
    <table class="default">
                            <tr>
                                <th scope="row">ITERACIÓN</th>
                                <th> PILA<- </th>
                                <th> ENTRADA </th>
                                <th> TRANSICIONES </th>
                            </tr>"""
    no_tk = """              <tr>
                                <th>%s</th>"""
    tk_fila =              """ <td>%s</td>"""
    fin_fila=                   """ </tr>"""
    fin      =             """ </table></center>"""

    contenido = encabezados

    for tk in iteraciones:
        if tk.id!="&" and tk.id != "#":
            contenido+= no_tk % (str(tk.id))
            contenido+= tk_fila % (str(tk.pila))
            contenido+= tk_fila % (str(tk.entrada))
            contenido+= tk_fila % (str(tk.transicion))
            contenido+= fin_fila
    contenido += fin
    contenido += "<h3>"+iteraciones[-1].entrada+"</h3>"

    return contenido
```

Anexos:

1.Cargar Archivo de entrada:



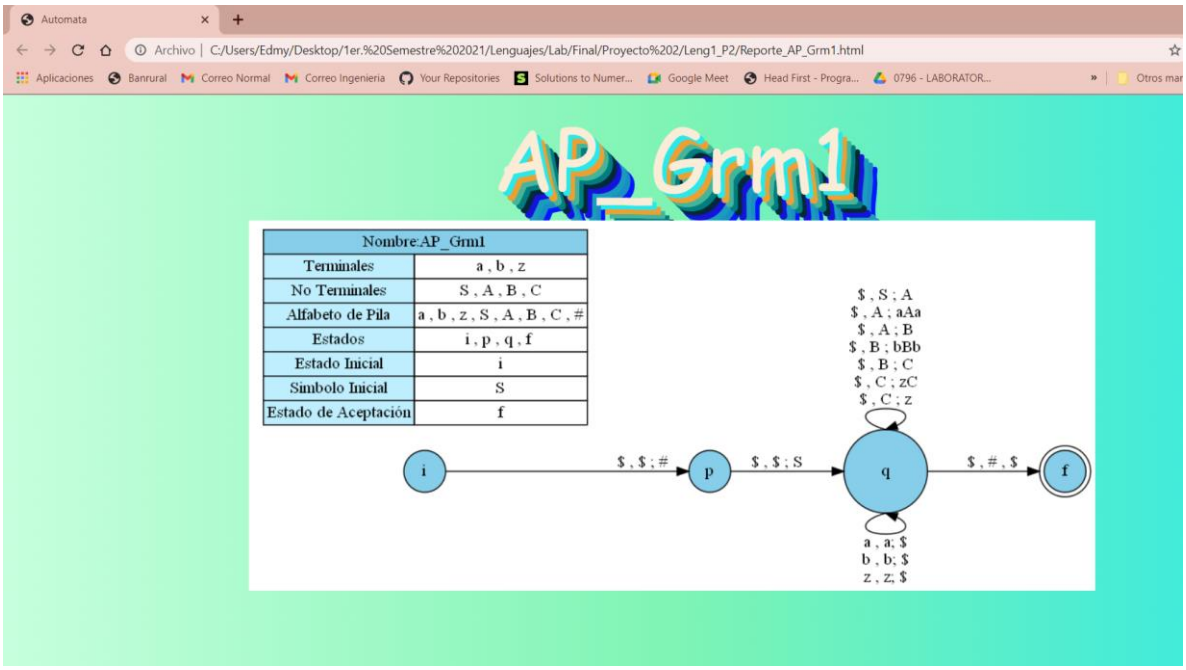
2. Mostrar información general de la gramática

```
Estas son las Gramaticas Cargadas en Memoria:
1. Gramatica1
2. Grm1

Ingrese el numero correspondiente a su elección:
2
```

```
Nombre de la Gramatica tipo 2: Grm1
No terminales: ['S', 'A', 'B', 'C']
Terminales: ['a', 'b', 'z']
No terminal Inicial:  S
Producciones:
S->A
A->aAa | B
B->bBb | C
C->zC | z
```

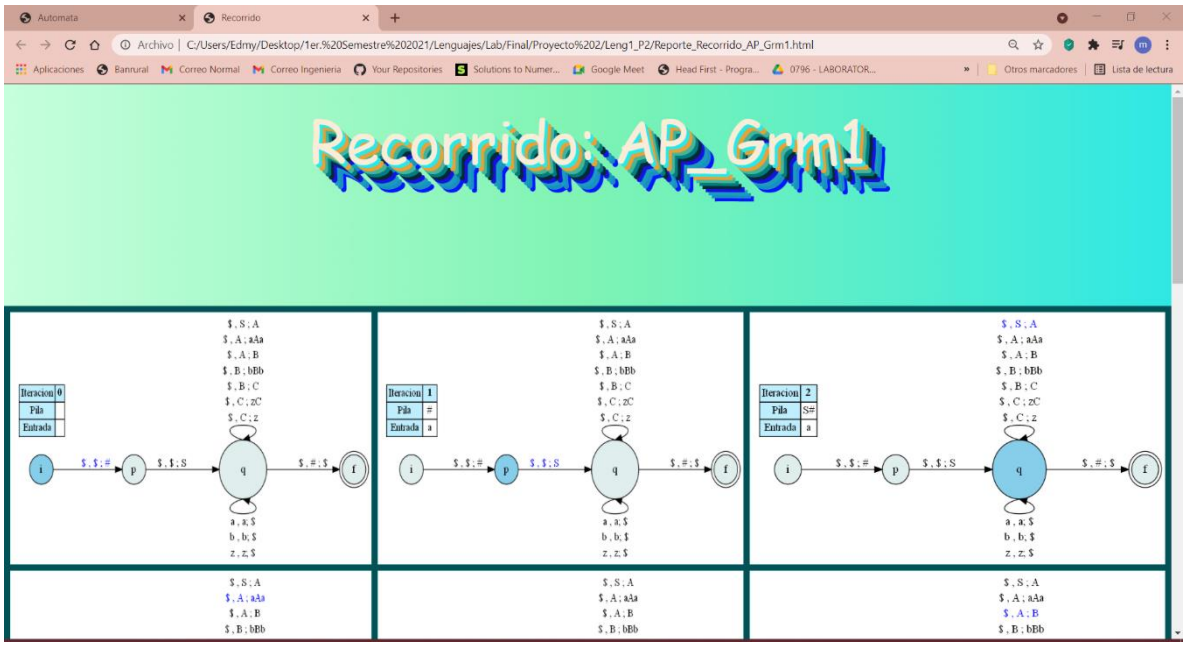
3. Generar autómata de pila equivalente

```
Estas son las Gramaticas Cargadas en Memoria:
1. Gramatica1
2. Grm1

Ingrese el numero correspondiente a su elección:
2
```



4. Reporte de recorrido

```
Estas son los Automatas de Pila Cargadas en Memoria:
1. AP_Gramatica1
2. AP_Grm1

Ingrese el numero correspondiente a su elección:
2
Introduzca la palabra a evaluar
abzba
```



5. Reporte en Tabla:

ITERACIÓN	PILA<-	ENTRADA	TRANSICIONES
0		a	(i,$,$;p,#)
1	#	a	(p,$,$;q,S)
2	S#	a	(q,$,S;q,A)
3	A#	a	(q,$,a;q,aAa)
4	aAa#	a	(q,a,a;q,$)
5	Aa#	b	(q,$,B;q,B)
6	Ba#	b	(q,$,b;q,bBb)
7	bBba#	b	(q,b,b;q,$)
8	Bba#	z	(q,$,C;q,C)
9	Cba#	z	(q,$,z;q,z)
10	zba#	z	(q,z,z;q,$)
11	ba#	b	(q,b,b;q,$)
12	a#	a	(q,a,a;q,$)
13	#	$	(q,$,#;f,$)
14	$	$	f