

AP

Automáta de Pila

MANUAL TÉCNICO

Contenido

INTRODUCCIÓN:.....	3
REQUERIMIENTOS DEL SISTEMA	4
Windows:.....	4
Mac OS X:.....	4
Linux:	4
Menú	5
Cargar archivo	6
Generar reporte de tablas:	7
AFD	8
AP	8
PARADIGMAS DE PROGRAMACIÓN UTILIZADAS	10
Programación estructurada	10
Programación procedimental	11
Programación modular.....	12
Programación Orientada a Objetos	12
.....	12
Estructura y formato de los archivos	13
CONCLUSIÓN:.....	14

INTRODUCCIÓN:

Se realizó una aplicación, que permite analizar y procesar información de un archivo de texto dado con un formato, el cual contiene una estructura y en base a esa estructura se identifican los diferentes patrones para extraer la información que contiene y con eso se generan autómatas de pila y también se generan reportes que permite visualizar los procedimientos, se hizo a través del lenguaje Python, se hizo uso de HTML y CSS para generar reportes web y también se hizo uso de librerías externas como Tkinter para mostrar un explorador de archivos.

REQUERIMIENTOS DEL SISTEMA

Windows:

- Python 3.8.1
- Windows 10 (8u51 y superiores)
- Windows 8.x (escritorio)
- Windows 7 SP1 • Windows Vista SP2

Mac OS X:

- Python 3.8.1
- Mac con Intel que ejecuta Mac OS X 10.8.3+, 10.9+
- Privilegios de administrador para la instalación

Linux:

- Python 3.8.1
- Cualquier distribución de Linux.

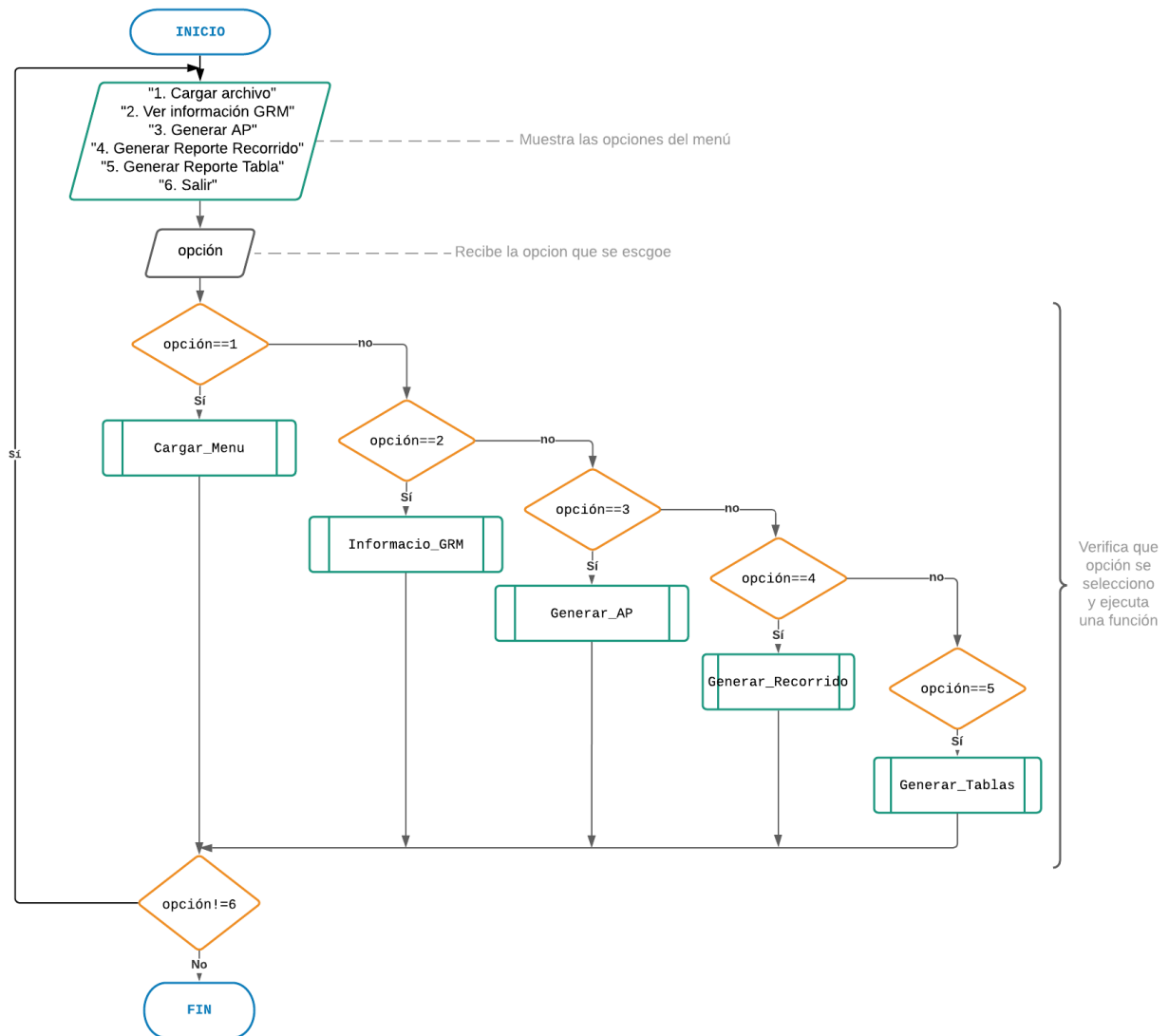
En cualquiera de los sistemas que vaya a ejecutar la aplicación se debe contar con un navegador que soporte HTML 5, para poder visualizar el reporte.

- Python en la versión 3.8.1 en adelante, puede descargarlo si no lo tiene desde su pagina oficial: <https://www.python.org/downloads/>.
- Graphviz puede descargarlo desde su pagina oficial: <https://www.graphviz.org/download/>.
- Paquete de python graphviz, si no se tiene este paquete puede instalarlo ejecutando el siguiente comando desde una terminal.

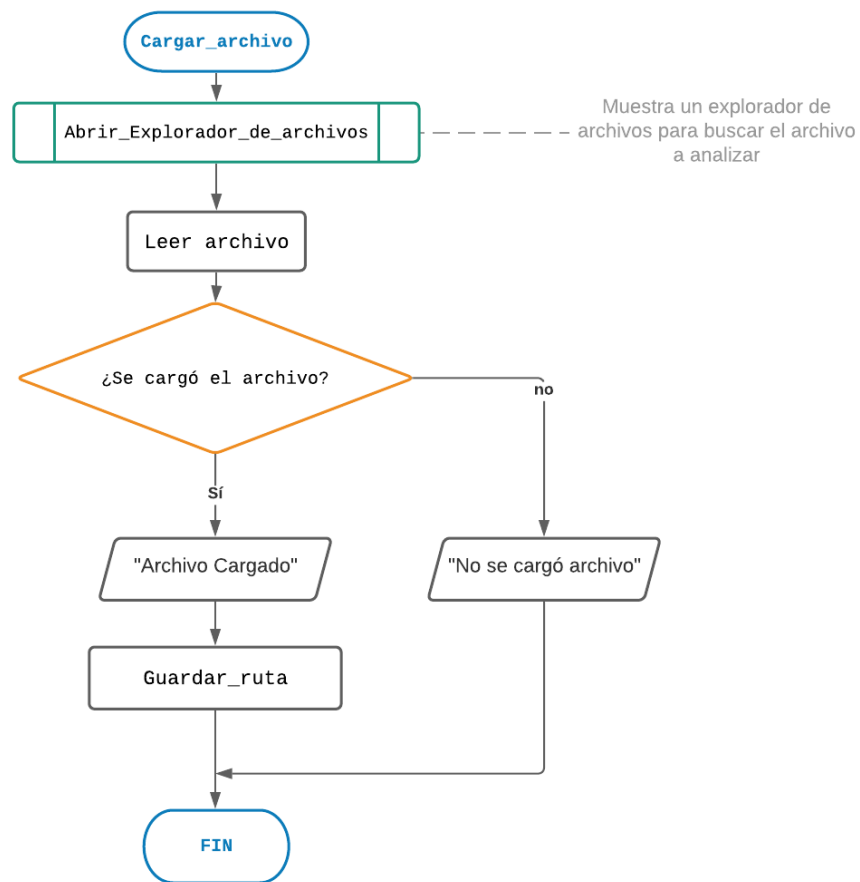
A screenshot of a terminal window with a dark background and light blue text. The command 'pip install graphviz' is entered. The window has three colored dots (red, yellow, green) in the top left corner, typical of a macOS window.

```
pip install graphviz
```

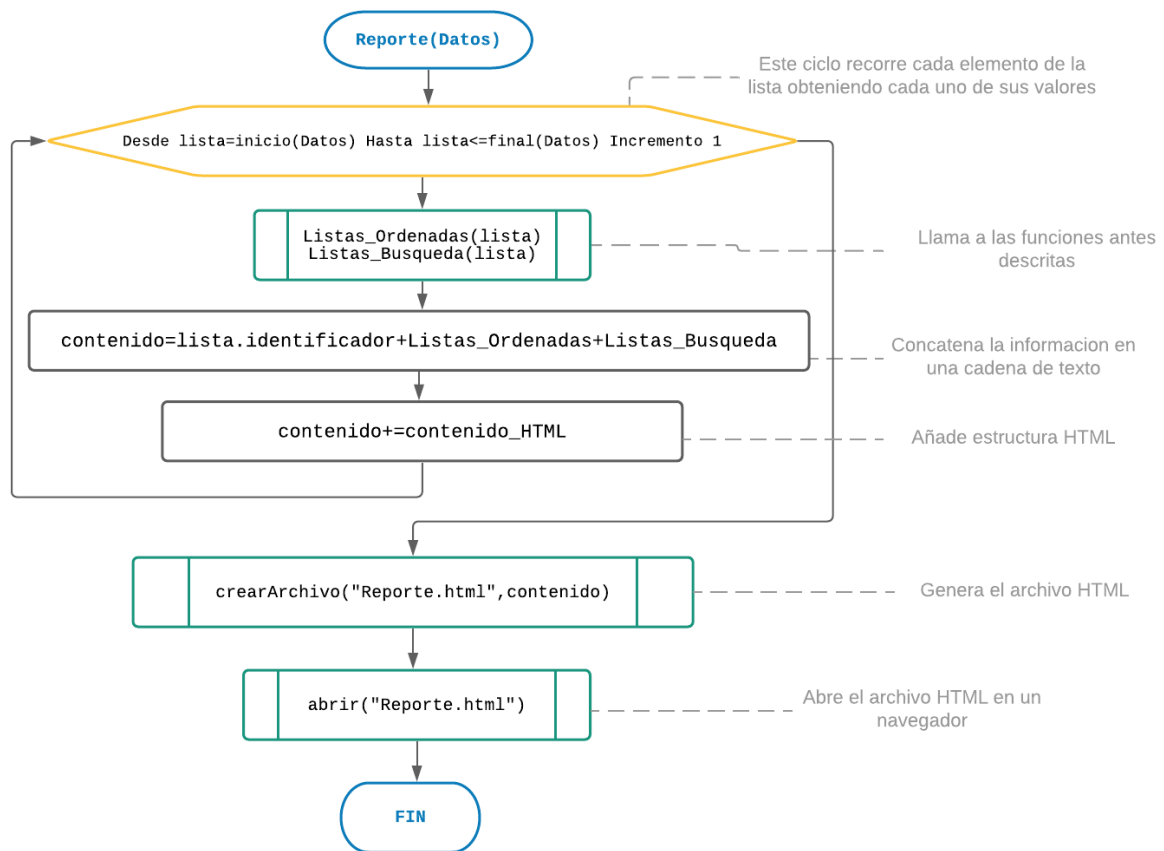
Menú



Cargar archivo

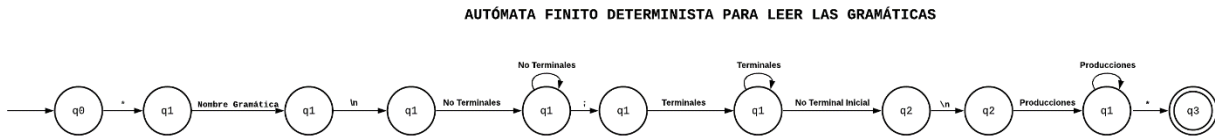


Generar reporte de tablas:



AFD

AUTÓMATA FINITO DETERMINISTA PARA LEER LAS GRAMÁTICAS

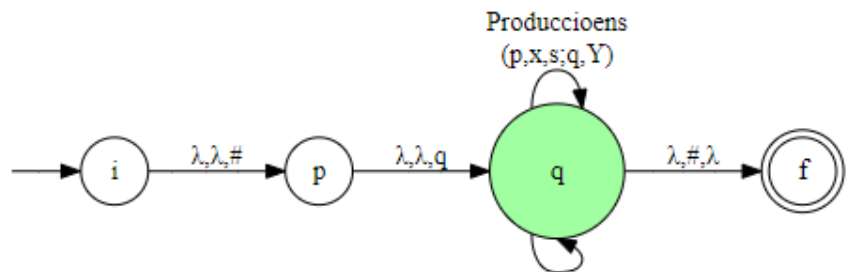


Este autómata permite, la lectura de las gramáticas contenidas en el archivo obteniendo cada una de sus características y propiedades para poder generar el autómata de pila.

AP

- Un autómata de pila es una séxtupla de la forma $(S, \Sigma, \Gamma, \delta, lo, F)$ donde:
- - S es un conjunto finito de estados
- - Σ es el alfabeto del autómata de pila. $\Sigma = T \cup N$
- - Γ es el conjunto finito de símbolos de pila
- - δ es el conjunto de transiciones o cambios de estado
- - lo es el estado inicial
- - F es el conjunto de estados de aceptación

Terminales = { }
 Alfabeto de pila = { }
 Estados = { i, p, q, f }
 Estado inicial = { i }
 Estado de aceptación = { f }



- Este proceso se representa con la notación $(p, x, s; q, Y)$, donde:
- p es el estado actual
- x es el símbolo de entrada
- s es el símbolo que en la cima de la pila
- q es el nuevo estado
- Y es el símbolo que se inserta en la pila

El autómata de pila consta de cuatro estados definidos { i, p, q, f }, donde i es el estado inicial, q es el estado que evalúa todas las producciones y las va apilando conforme su función sea establecida y f es el estado de aceptación, este autómata de pila valida las cadenas por el método de pila vacía.

PARADIGMAS DE PROGRAMACIÓN UTILIZADAS

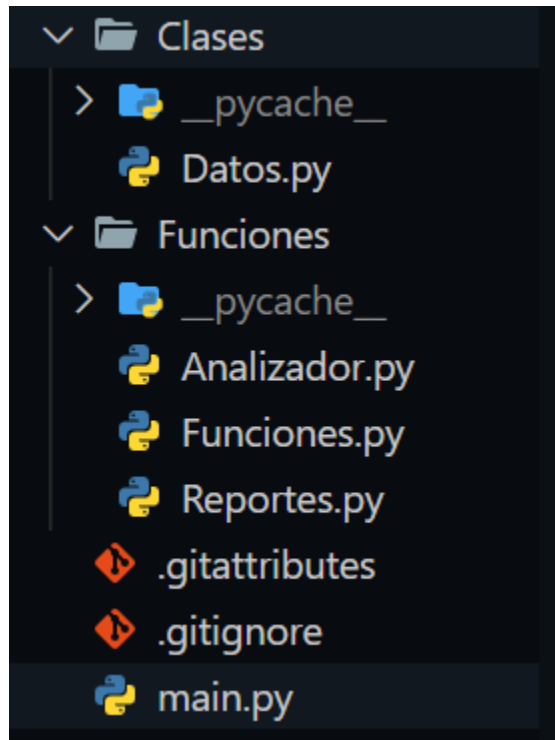
Programación estructurada

```
LeerArchivo.py x
LeerArchivo.py
22
23     for i in range(2,len(lineas)):
24         produccion=lineas[i].split("->")
25         Expresiones=produccion[1].lstrip().rstrip()
26         if len(Expresiones.split()) > 2:
27             agregar=True
28
29     if agregar:
30         for i in range(2,len(lineas)):
31             listaAux=[]
32             produccion=lineas[i].split("->")
33             NoT=produccion[0]
34             Expresiones=produccion[1].lstrip().rstrip()
35             listaAux.append(Expresiones)
36             if len(listaProduccion) Loading...
37                 listaProducciones.append(exp(NoT,listaAux))
38             else:
39                 encontrado=False
40                 pos=None
41                 for buscar in listaProducciones:
42                     if buscar.NoT == NoT:
43                         encontrado=True
44                         pos=listaProducciones.index(buscar)
45                         #print(pos)
46                         break
47
48             if encontrado:
49                 listaProducciones[pos].expresiones.append(Expresiones)
50             else:
51                 listaProducciones.append(exp(NoT,listaAux))
52         gramaticas.append(GRM(nombre,NT,T,TT,listaProducciones))
53
```

Programación procedimental

```
ValidarCadena.py X
Funciones > ValidarCadena.py
4
5 class validar:
6     def __init__(self, gramatica, cadena):
7         self.pila=[]
8         self.gramatica=gramatica
9         self.historial=[]
10        self.transionesNTerminales=[]
11        self.cadena=cadena
12        self.generarTransiciones()
13        self.verificar()
14
15    def generarTransiciones(self):
16        for i in self.gramatica.P:
17            for j in i.expresiones:
18                self.transionesNTerminales.append(TransicionNT(i.NoT,j))
19            '''
20        for i in self.transionesNTerminales:
21            print(i)
22        '''
23
24    def verificar(self):
25        cont=2
26        if self.verificarAlfabeto(self.cadena[0]):
27            self.historial.append(historial(0,"",self.cadena[0],"(i,λ,λ;p,#)")
28            self.pila.append("#")
29            self.historial.append(historial(1,self.verPila(),self.cadena[0],"(p,λ,λ;q,"
30            self.pila.append(self.gramatica.NTI)
31            longitud=len(self.cadena)
32            posicion=0
33            while posicion<longitud:
34                ton=self.obtenerTon()
```

Programación modular



```
from Funciones import Analizador
from Funciones import Funciones
from Funciones import Reportes
```

Programación Orientada a Objetos

A screenshot of a code editor window titled 'Gramatica.py'. The code defines a class `GRM` and an object `exp`. The `GRM` class has an `__init__` method that initializes `nombre`, `NT`, `T`, `NTI`, and `P`. It also has a `__str__` method that returns a string representation of the grammar. The `exp` object has an `__init__` method that initializes `NoT` and `expresiones`. It also has a `__str__` method that returns a string representation of the object.

```
1 class GRM:
2     def __init__(self, nombre, NT, T, NTI, P):
3         self.nombre=nombre
4         self.NT=NT
5         self.T=T
6         self.NTI=NTI
7         self.P=P
8
9     def __str__(self):
10        string="- Nombre de la gramática tipo 2: "+str(self.nombre)+"\n- No Terminales: = { "+str(self.NT)+" ;
11        return string
12
13 class exp:
14     def __init__(self, NoT, expresiones):
15         self.NoT=NoT
16         self.expresiones=expresiones
17
18     def __str__(self):
19        string=" "+str(self.NoT)+str(" -> ")
20        for i in range(len(self.expresiones)):
21            if i!= len(self.expresiones)-1:
22                string+=str(self.expresiones[i])+" / "
23            else:
24                string+=str(self.expresiones[i])
25        return string
```

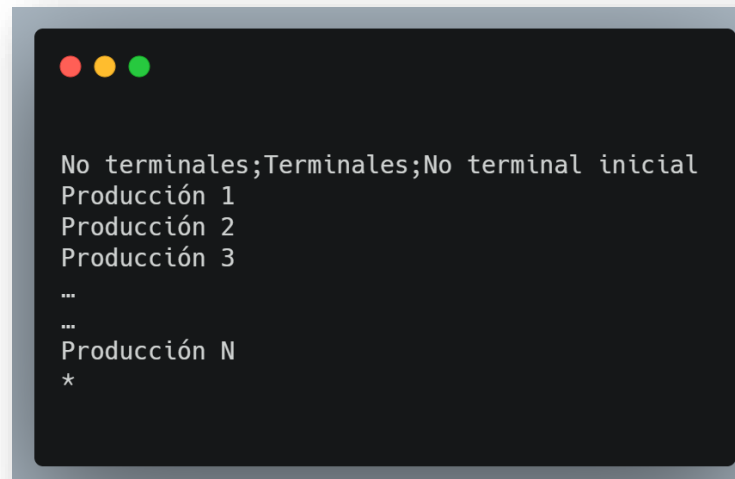
Estructura y formato de los archivos

Los archivos deberán ser guardados con la extensión `.glc`

Dentro del archivo de entrada podrán venir N gramáticas, el final de cada gramática estará marcado por un asterisco (*).

Las partes de cada gramática estarán especificadas en el orden siguiente:

Para el caso de los no terminales y terminales vendrán separados por comas.



```
No terminales;Terminales;No terminal inicial
Producción 1
Producción 2
Producción 3
...
...
Producción N
*
```

Ejemplo de archivo de entrada:

		entrada.glc
Nombre	1	Grml
No Terminales;Terminales;Terminal inicial	2	S,A,B,C;a,b,z;S
Producción 1	3	S->A
Producción 2	4	A->a A a
Producción 3	5	A->B
Producción 4	6	B->b B b
Producción 5	7	B->C
...	8	C->z C
Producción n	9	C->z
Fin de la gramática	10	*
Nombre	11	Gramatica2
No Terminales;Terminales;Terminal inicial	12	A,B,C,D;0,1;A
Producción 1	13	A->1 B
Producción 2	14	A->0 C
Producción 3	15	B->1 A
Producción 4	16	B->0 D
Producción 5	17	C->1 D
Producción 6	18	C->0
...	19	D->1 C
...	20	C->0 A
Producción n	21	D->0 B
Fin de la gramática	22	*
	23	
	24	
	...	
	...	

CONCLUSIÓN:

- Es necesario establecer un formato y estructura al archivo que se desea analizar y procesar ya que con ello se pueden crear algoritmos que ayudaran a realizar tales acciones.
- El explorador de archivos hace más fácil la búsqueda y carga de archivos.
- Los reportes permiten visualizar de forma detallada cierta información.