
MANUAL TÉCNICO

# Proyecto 2

**LFP-1S**

2021

201904013
Marlon Isaí Figueroa Farfán

# DESCRIPCIÓN GENERAL

Esta es una aplicación de escritorio desarrollada con el lenguaje de programación Python versión 3.8.1, utilizando como IDE Visual Studio Code.

Esta aplicación posee un menú de funcionalidades capaz de cargar archivos, leerlos, para posteriormente realizar un análisis, generar gramáticas y autómatas de pila mostrando los grafos en una página desarrollada con HTML implementando CSS y BOOSTRAP.

# REQUERIMIENTOS DEL SISTEMA

SISTEMA OPERATIVO:

- MS Windows XP o superior.
- Apple OSX 10.4.x o superior.
- GNU/Linux 2.6.x o superior.

PROCESADOR:

- Procesador de 1,6 GHz o superior.

MEMORIA:

- 1 GB de RAM (1,5 GB si se ejecuta en una máquina virtual)

ESPACIO EN DISCO:

- 20 GB de espacio disponible en disco duro.
- Disco duro de 5400 RPM.

# CLASES

Librerías que se deben importar para el funcionamiento correcto de la aplicación:

```
import os
import time
import sys
import tkinter.filedialog
import webbrowser
from Gramatica import GLC
from Gramatica import GRAMA
```

Para el desarrollo de la aplicación se utilizó POO creando dos clases:

**Clase GLC**: tiene la estructura para almacenar los datos de las gramáticas leídas del archivo cargado por el usuario.

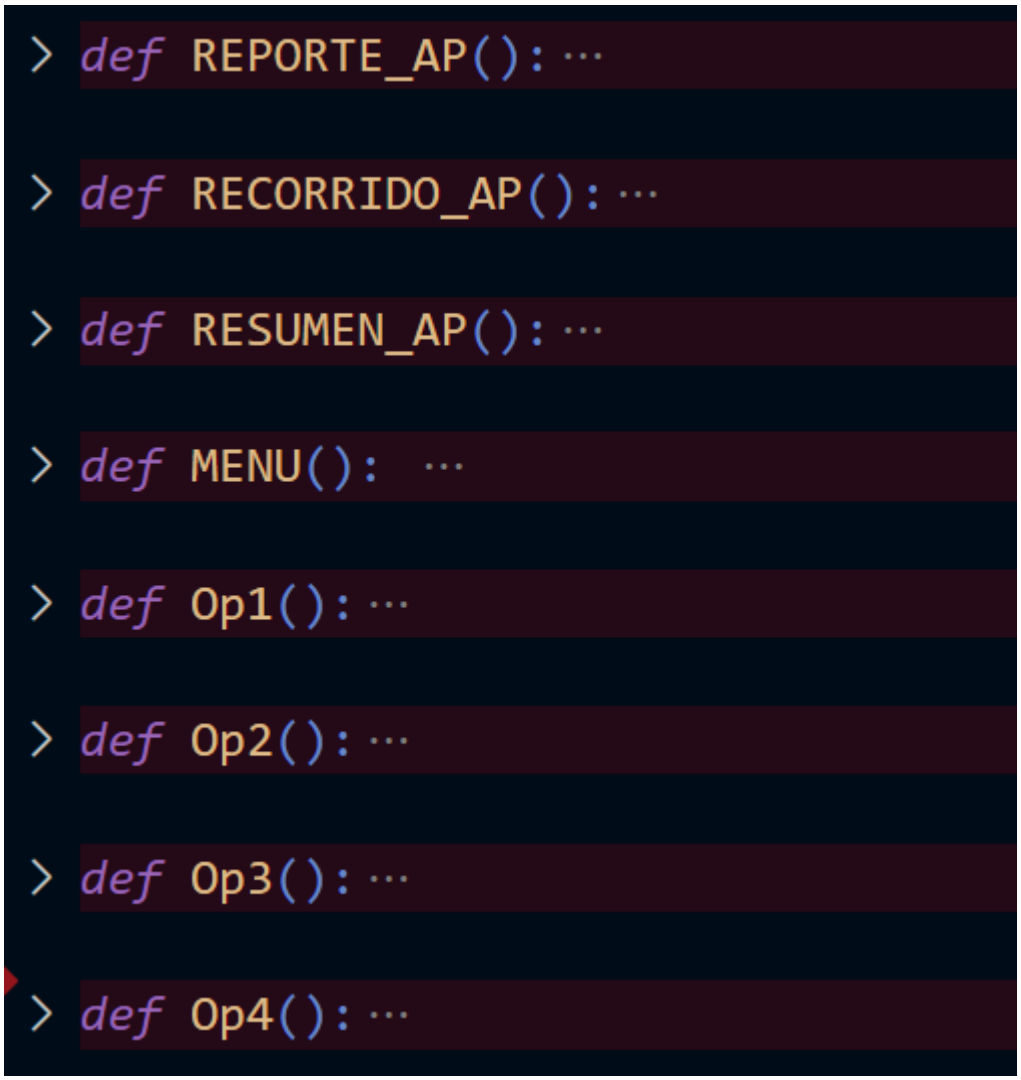
```
class GLC():
    def __init__(self,nombre,no_terminales,terminales,inicial,transiciones):
        self.nombre=nombre
        self.no_terminales=no_terminales
        self.terminales=terminales
        self.inicial= inicial
        self.transiciones=transiciones
```

**Clase GRAMA**: tiene la estructura para almacenar los datos necesarios de las producciones leídas del archivo cargado por el usuario para generar la nueva gramática.

```
class GRAMA():
    def __init__(self,no_terminales,transiciones):
        self.no_terminales=no_terminales
        self.transiciones=transiciones
```

# MÉTODOS

Esta es la vista general de los métodos utilizados para el desarrollo de la aplicación:




En la siguiente sección se detallan las funcionalidades esenciales de los métodos creados para el desarrollo de la aplicación.

# FUNCIONALIDADES

**MENÚ:** En este método se muestra al usuario el menú de opciones con el que cuenta la aplicación y en base a este se mandan a llamar a los otros métodos.

```
def MENU():
    os.system('cls')
    print("_____________________________________________________________________________")
    print("
    print ("        \t              1 - Cargar Archivo           ")
    print ("        \t              2 - Mostrar Información          ")
    print ("        \t              3 - Generar Autómata de Pila          ")
    print ("        \t              4 - Reporte de Recorrido       ")
    print ("        \t              5 - Salir")
    print("_____________________________________________________________________________")
```

**OP3():** En el método para generar el autómata de pila equivalente se crean las producciones necesarias para obtener un grafo correcto de la gramática seleccionada por el usuario.

```
#separando derecho
print(produccion)
m="λ,λ;#"
s="λ,λ;"+i.inicial
q="λ,#;λ"
trans=[]
for x in range(len(produccion)):
    a=produccion[x][0]
    b=produccion[x][1]
    c=("λ,"+str(a.replace(" ",""))+";"+str(b.replace(" ","")))
    trans.append(c)
for t in range(len(i.terminales)):
    a=i.terminales[t]
    y=a+","+a+";λ"
    trans.append(y)
cadena=""
for p in trans:
    cadena=cadena + str(p)+"\n"
```

**OP1():** En el método de la lectura de archivo se leen las producciones para posteriormente analizarlas y determinar si la gramática es regular o libre de contexto, si esta es libre de contexto se almacena en la memoria del sistema, si no se guarda en otro espacio para posteriormente generar un reporte.

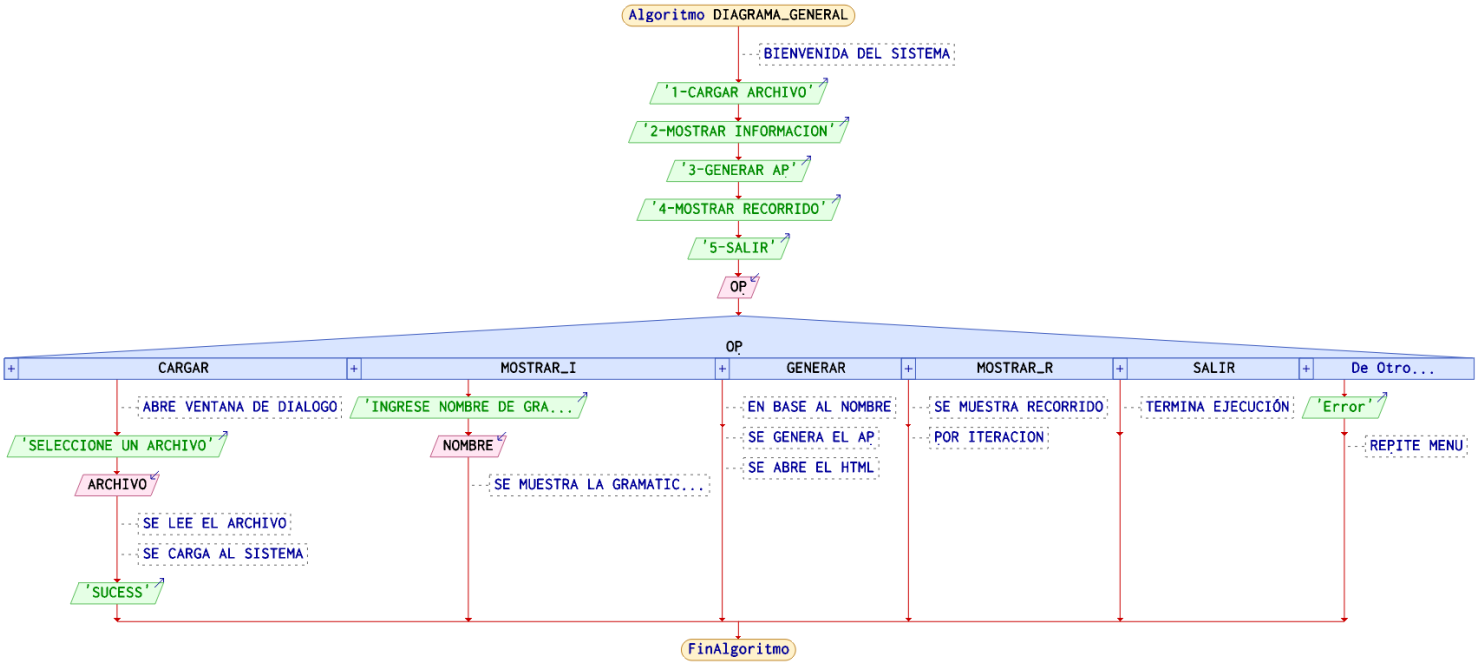
```
for x in range(len(transiciones)):
    produccion.append(transiciones[x].split("->"))
cont+=1
analisis=[]
for i in range(len(produccion)):
    analisis.append(produccion[i][1].split(" "))
es_libre=False
for i in analisis:
    if len(i)==3 or len(i)>3:
        es_libre=True
#print("transiciones :"+ str(transiciones))
if es_libre==True:
    GRAMATICAS.append(GLC(nombre,no_terminales,terminales,inicial,transiciones))
else:
    REPORTE.append(GLC(nombre,no_terminales,terminales,inicial,transiciones))
```

**OP2():** En el método de mostrar información en base a un nombre se analizan las producciones para generar una gramática libre de contexto y mostrarla al usuario.

```
print("Producciones")
for x in range(len(i.transiciones)):
    produccion.append(i.transiciones[x].split("->"))
o=0
armando=[]

while o<len(i.no_terminales):
    trans=[]
    for x in range(len(produccion)):
        if produccion[x][0]==i.no_terminales[o]:
            trans.append(produccion[x][1])
    armando.append(GRAMA(i.no_terminales[o],trans))
    o+=1
```

# DIAGRAMAS




# ANEXOS

Dada una gramática G independiente del contexto es posible construir un autómata de pila M de la manera siguiente:

1. ***Designe el alfabeto del autómata M*** como los ***símbolos terminales de G***, y los ***símbolos de pila de M*** como los ***símbolos terminales y no terminales de G,*** junto con el símbolo especial **#**.
2. Designe los estados del autómata **M** como ***i,p,q,f*** donde ***i*** es el estado inicial y ***f*** es el único estado de aceptación.
3. Introduzca la ***transición (i, λ, λ;p,#)***
4. Introduzca la transición ***(p, λ, λ;q,S)*** donde ***S es el símbolo inicial de G.***
5. Introduzca una transición de la forma ***(q,λ,N;q,w)*** para cada regla de reescritura N→w en G.
6. Introduzca una transición de la forma ***(q,x,x;q,λ)*** para cada terminal de x de G (es decir, para cada símbolo del alfabeto de M).
7. Introduzca la transición ***(q, λ,#;f, λ)***

*"Teoría de la computación" de Glenn Brooksheer página 85*

Lógica utilizada para generar el autómata de pila:



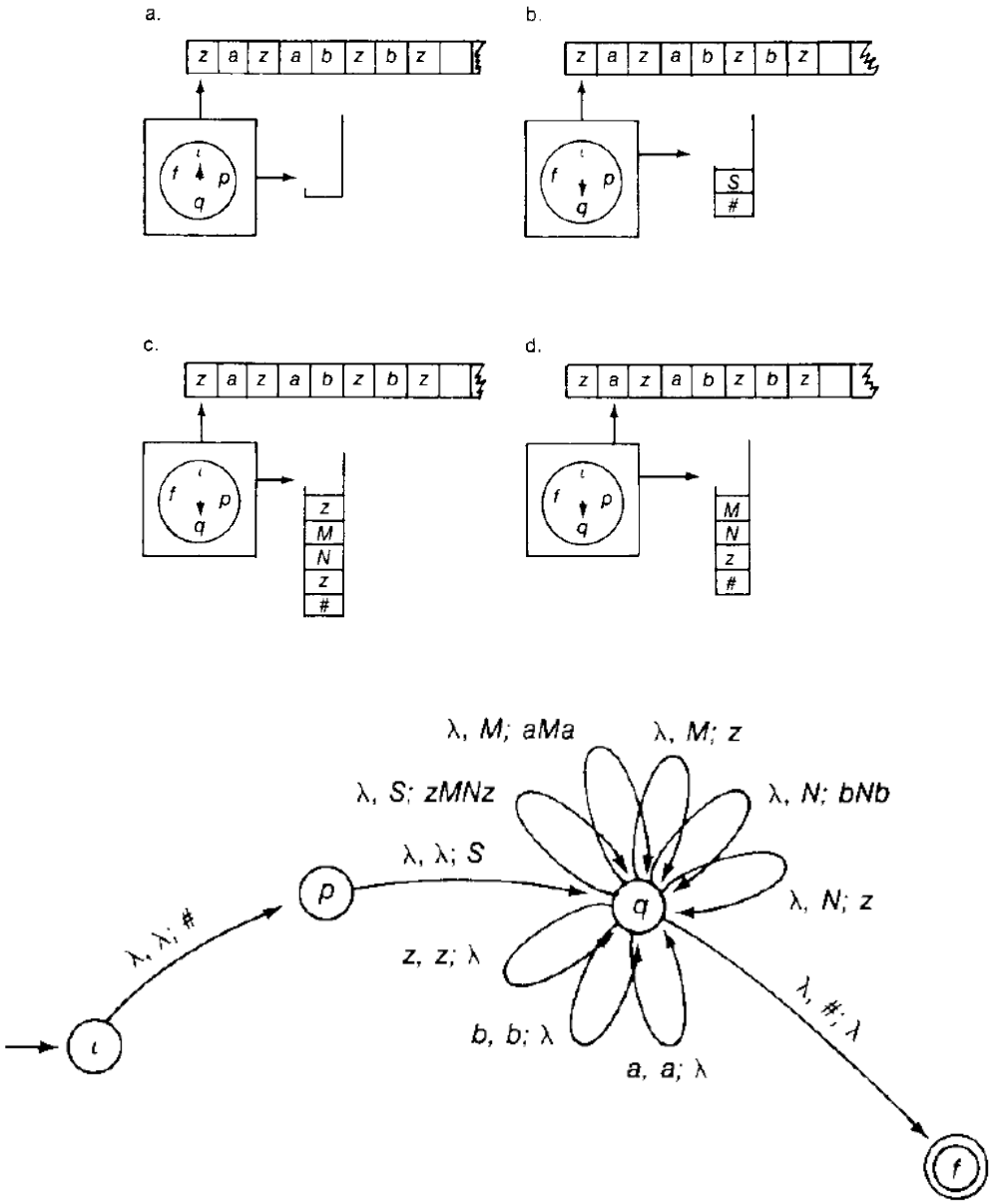



**Figura 2.8** Diagrama de transiciones de un autómata de pila construido a partir de la gramática de la figura 2.5 utilizando las técnicas presentadas en la demostración del teorema 2.2