­

B

A

PROYECTO 2

Lenguajes Formales y de Programación

Manual Técnico

2021

Nombre: Gerson Sebastian Quintana Berganza

Carné: 201908686

# DESCRIPCIÓN DE LA APLICACIÓN

La aplicación fue desarrollada utilizando como lenguaje de programación *Python* y utilizando la herramienta *Graphviz* para la generación de imágenes, la cual tiene como objetivo principal profundizar acerca de los lenguajes independientes del contexto, el cual permite navegar entre diferentes opciones las cuales agrupan funciones de mucha utilidad para poder comprender mucho mejor el funcionamiento de los lenguajes libres del contexto.

El flujo de la aplicación va desde la lectura de un archivo de texto que contiene una determinada cantidad de gramáticas con cierta estructura, que con opciones de la aplicación se pueden ver desglosadas cada una de las gramáticas si así se desea. Además, con estas gramáticas ingresadas se pueden realizar un conjunto de operaciones que son el corazón de la aplicación, ya que dentro de una de las operaciones que se pueden realizar en la aplicación esta poder ingresar una cadena de texto que será reconocida por un determinado autómata y verificar si esta cadena es válida.

Además, la aplicación es capaz de generar reportes en html, estos reportes son: la representación gráfica de las iteraciones realizadas por un autómata para reconocer o no una cadena de texto; la representación gráfica de una tabla que muestra las iteraciones realizadas por el autómata para validar o no la cadena; y no reporte que muestra todas las gramáticas que no fueron cargadas a la aplicación por no ser exclusivamente independientes del contexto.

## DATOS TÉCNICOS

Lenguaje de programación utilizado: Python 3.8.0

IDE utilizando: Visual Studio Code 1.48.2

Sistema operativo: Windows 10 (64 bits)

Lenguaje utilizado para la generación de reportes: HTML

Herramienta utilizada para generar grafos: Graphviz

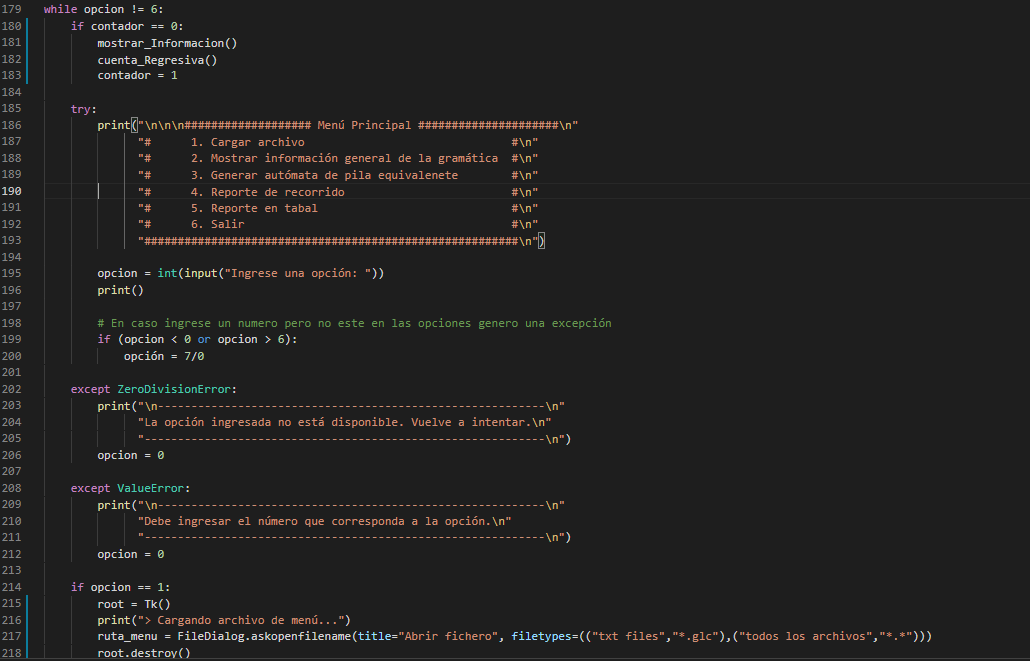
# LÓGICA DEL PROGRAMA

## FUNCIONES UTILIZADAS EN LOS MÓDULOS

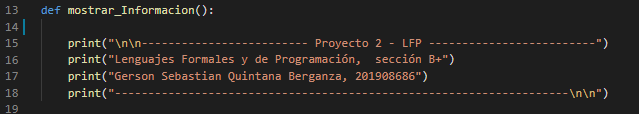
* **Main.py**

Este módulo es el encargado de controlar el flujo de todo el programa, ya que es en donde se eligen las opciones del menú y se manejan diferentes excepciones.

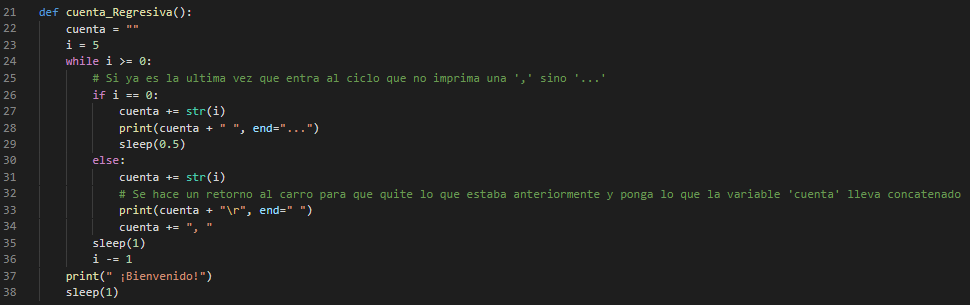
Lo primero que se ejecutará en este módulo cuando el usuario ingrese será una estructura repetitiva while, la cual se ejecuta siempre y cuando el usuario no ingrese la *opción 6*. En cada ejecución se mostrará el menú de opciones.



* Al inicio de este ciclo y cuando la variable *contador* sea cero se ejecutarán las funciones *mostrar\_Informacion() y cuenta\_Regresiva().*
* **mostrar\_informacion():** Muestra la información de quien desarrollo el programa.

****

* **cuenta\_Regresiva():** Consta de una estructura repetitiva que imprime los número del 1 al 5 pero de forma inversa. Para esto concatena cada uno de esos números en una variable, utilizando un retorno de carro para posicionarse de nuevo al inicio de la línea y nuevamente imprimir lo que había concatenado. Así mismo, utiliza el módulo *sleep* para dar un intervalo de tiempo antes de mostrar lo concatenado nuevamente. Finalizado el ciclo muestra la palabra *¡Bienvenido!.*



* Al seleccionar la *opción 1*, abre una ventana en donde se debe de seleccionar el archivo de texto con extensión *.glc* con todas las gramáticas. Además, se valida que si se haya seleccionado un archivo.

Si se llega a encontrar alguna gramática que no sea exclusivamente libre del contexto durante el análisis del archivo se mostrará en un reporte que podrá ser generado si ingresa la letra *s*. Dicho reporte se abrirá automáticamente.

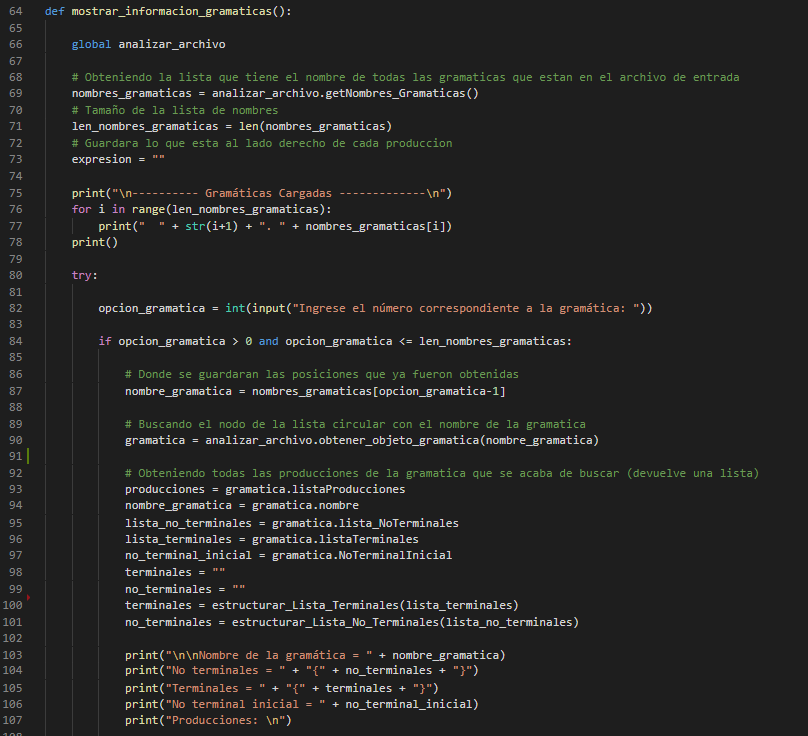


* Al seleccionar la *opción 2,* se ejecuta una función con nombre *mostrar\_informacion\_gramaticas()*.



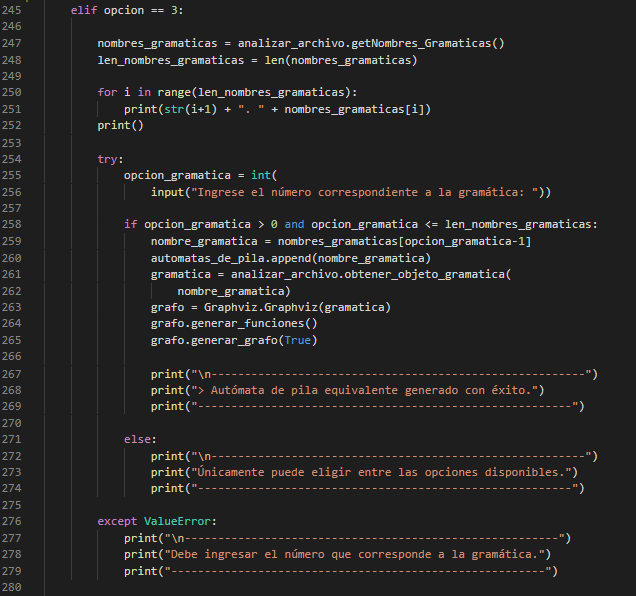
* **mostrar\_informacion\_gramaticas:** Obtiene el nombre de todas las gramáticas cargadas y almacenadas en el objeto *analizar\_archivo* y genera una estructura de modo que se muestren con la siguiente estructura:





* Al seleccionar la *opción 3,* obtiene el nombre de todas las gramáticas cargadas en el sistema y simplemente las muestra al usuario. Luego de mostrarlas le pide al usuario que ingrese el numero de la gramática de la cual desea generar el autómata de pila equivalente. Al ingresar el numero en realidad, lo que se va a enviar es nombre de la gramática que se encuentra en esa posición.

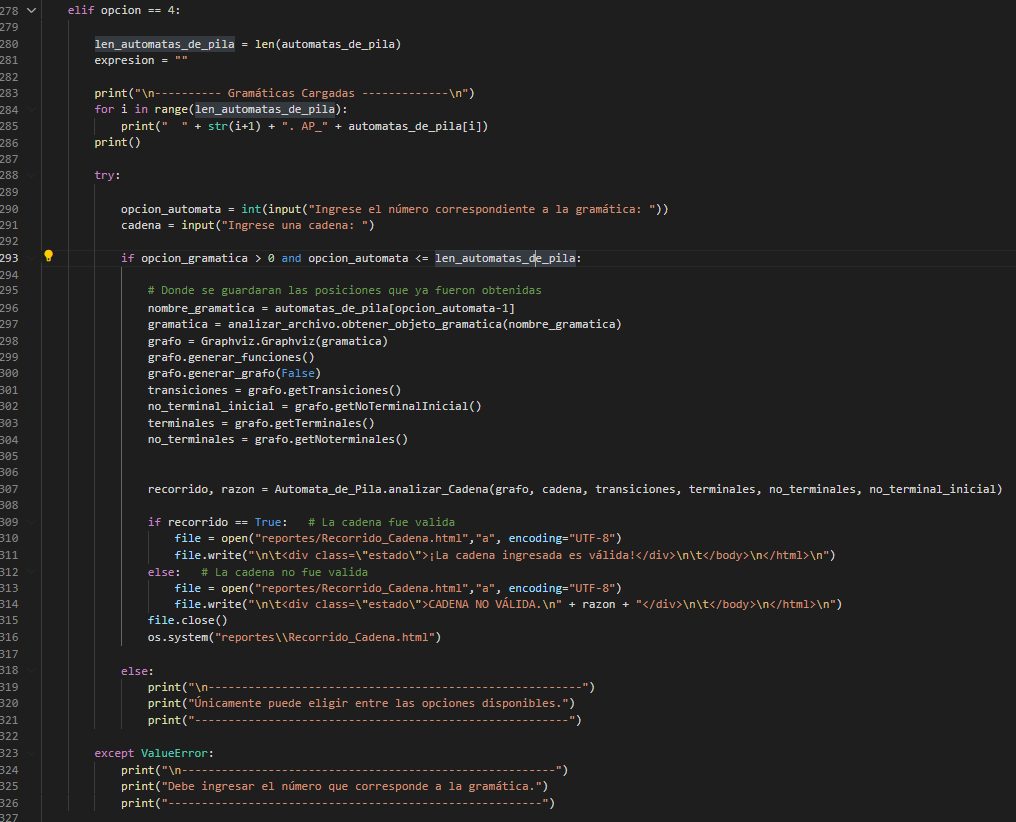
Una vez generado el autómata, el nombre de este se guarda en una lista con nombre *automatas\_de\_pila*. Asimismo se generan las funciones de transición de la forma *(q,$,A; q,aAa)*.



* Al seleccionar la *opción 4*, aquí toma la lista que guarda el nombre de todas las gramáticas de las cuales se generó un autómata de pila equivalente en la *opción 3* (*automatas\_de\_pila*) y muestra todos los nombres anteponiendo la cadena *AP\_*, generando asi un lista de todos los autómatas generados en la *opción 3*. Una vez elegido el automata, se le pide al usuario ingresar una cadena que será validada.

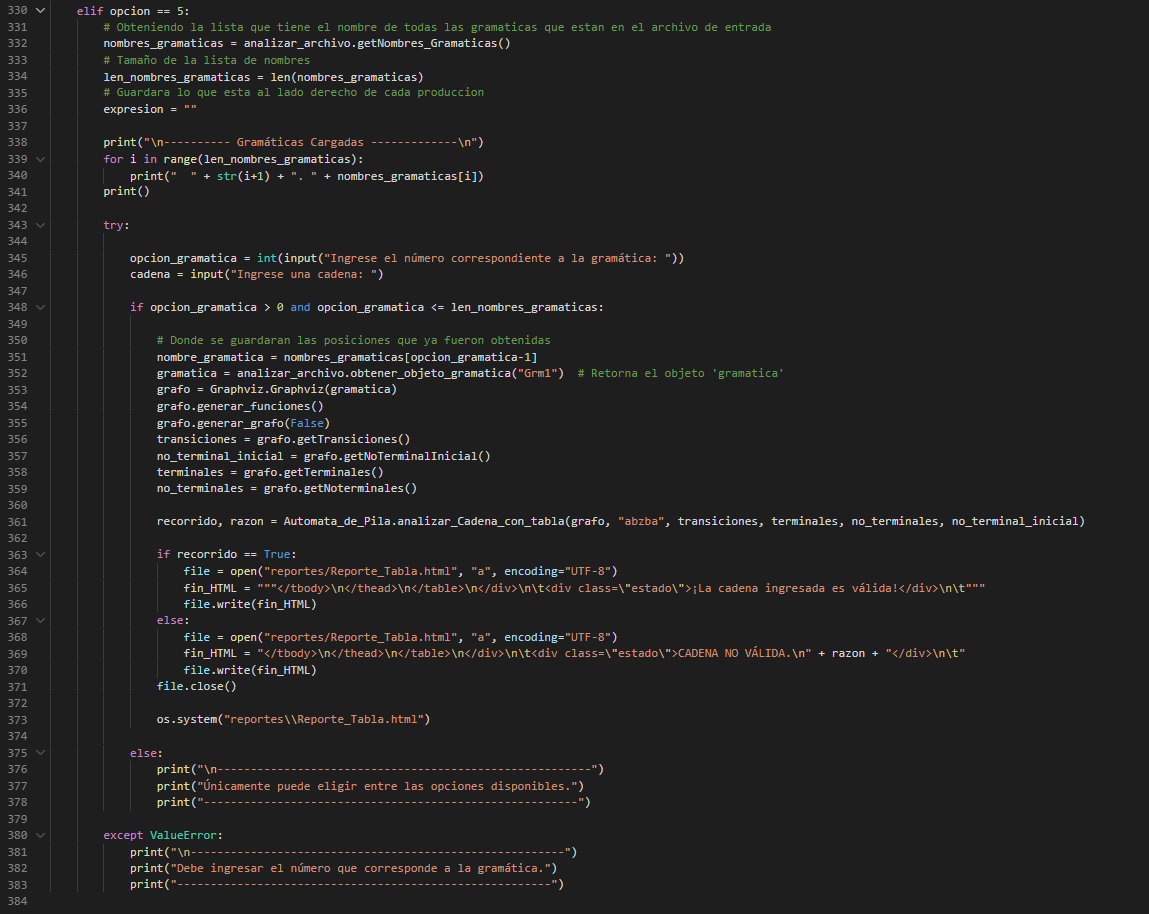
En la línea 297 obtiene el objeto *gramática* para que con la clase *Graphviz* generar las transiciones.

Una vez validada la cadena se genera un reporte html con todas las transiciones representadas con imágenes de grafos generados con *Graphviz*.



* Al seleccionar la *opción 5,* la lógica es básicamente la misma que en la *opción 4* ya que se muestran todos los nombres de los autómatas generados en la *opción 3* para que el usuario seleccione uno, ingrese una cadena de texto que será validada pero aquí en vez de generarse un reporte con todas las iteraciones, se muestra una tabla que muestra el número de iteración, la cadena en lectura y la transición utilizada.

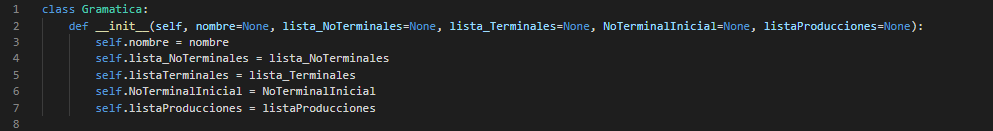
Una vez finalizado, abre automáticamente dicho reporte.



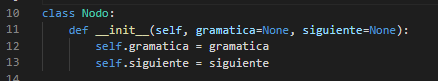
* La opción 6, lo que realiza es simplemente romper con el ciclo, haciendo que la condición que lo hace seguir ejecutándose ya no se cumpla.
* **Lista\_Cirular.py**

Modulo que implementa tres clases: *Gramatica, Nodo y Lista\_Circular*, en los cuales almacenará todas las partes necesarias de la gramática para realizar todas las operaciones posteriores.

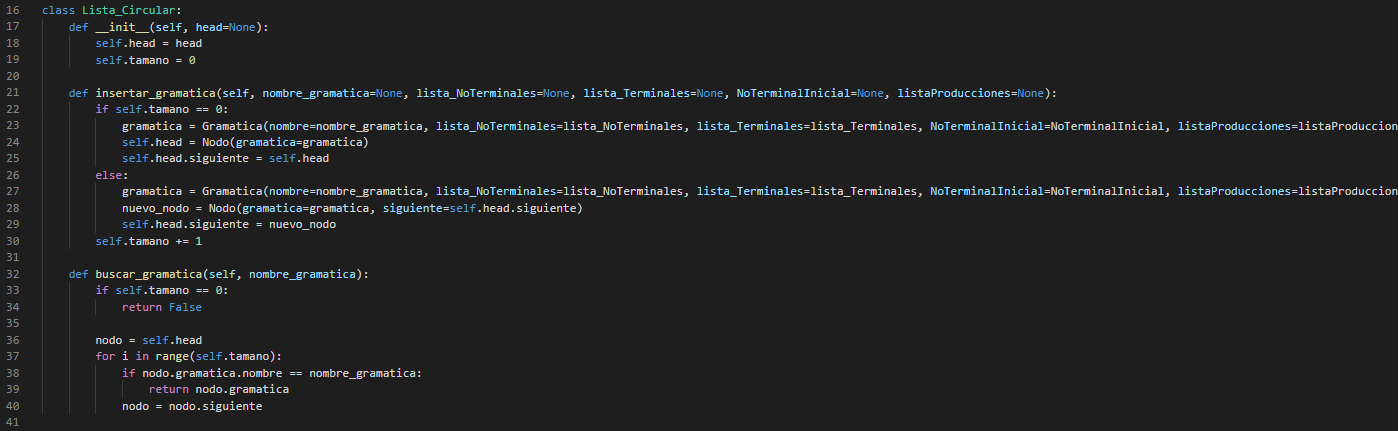
* **Clase Gramática:** Esta clase únicamente cuenta con su método constructor, que va a inicializar cada gramática libre del contexto. Este método recibe como parámetros, el nombre de la gramática, todos los no terminales de la gramática, todos los terminales de la gramática, el no terminal inicial y la lista de todas las producciones.



* **Clase Nodo:** Cuenta con el método constructor, el cual contendrá el objeto de la clase gramática, y un apuntador al siguiente nodo.



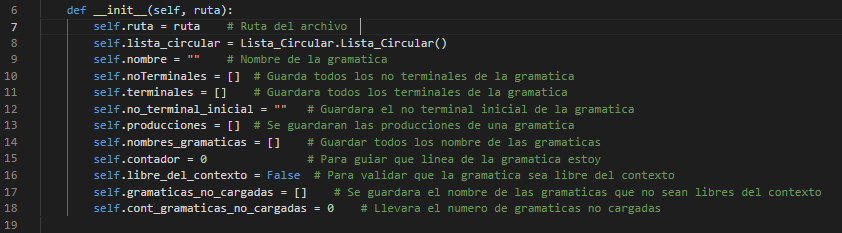
* **Clase Lista\_Circular:** Cuenta con tres funciones: *\_\_init\_\_()*, que inicializa la nueva cabecera de la lista circular y una variable *tamano*, que llevara la cuenta de cuantos nodos hay en la lista; *insertar\_gramatica(),* que recibe los mismos parámetros que el constructor de la clase Gramatica; *buscar\_gramatca()*, recibe como parámetro el nombre de la gramática que se quiere encontrar, retornando el objeto gramática que tiene ese nombre.



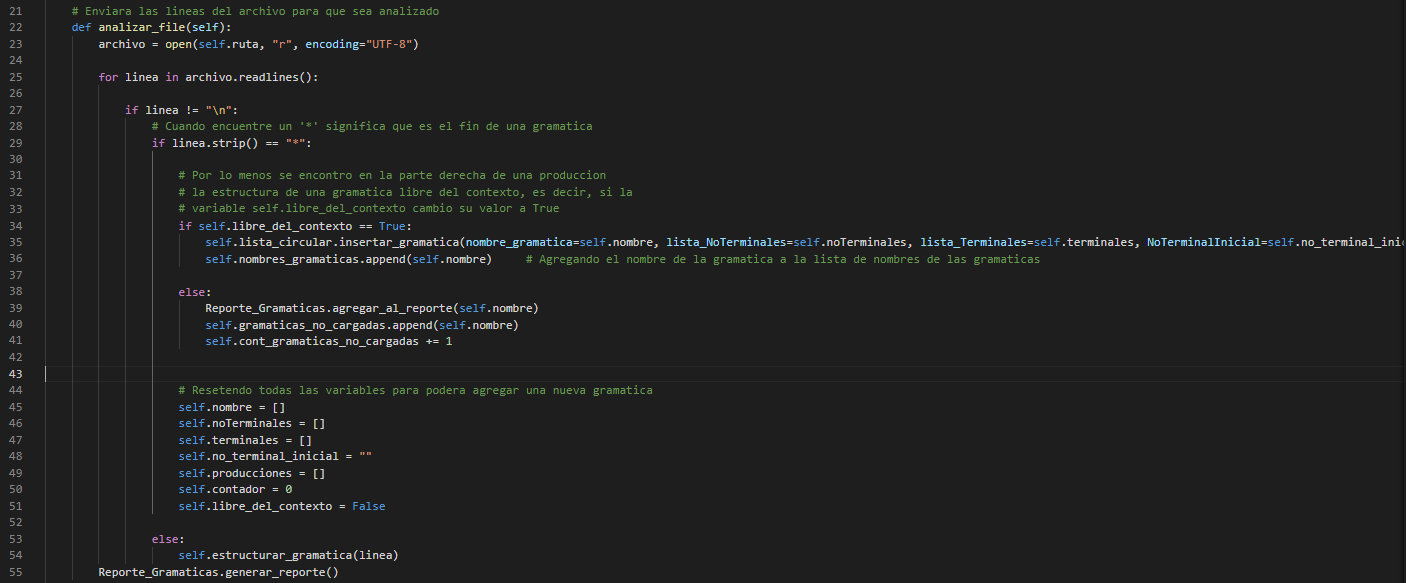
* **Analizar\_Archivo.py**

Es en este modulo donde se analiza la estructura del archivo de entrada y en base a ella almacenar los datos de tal modo que puedan ser fácilmente accedidos. Cuenta con las siguientes funciones:

* **\_\_init\_\_():** Recibe por parámetro la ruta del archivo que contiene a todas las gramáticas y es donde además se crea el objeto de la clase *Lista\_Circular*, en el cual se guardarán todas las gramáticas del archivo.



* **analizar\_file():** Abre el archivo del cual se recibió la ruta en el constructor y lo lee línea por línea siempre y cuando esta no este vacía. Lo que realiza durante este proceso es ir guardando en la lista *lista\_Producciones* todas las producciones de esa gramatica, en la *lista\_NoTerminales* todos los no terminales, en *lista\_Terminales* todos los terminales, y el nombre de la gramática. Esto lo hace hasta que encuentra un ‘\*’, cuando lo encuentra, inserta una nueva gramática en la lista circular y ‘resetea’ las variables utilizadas para que nuevamente vuelva a realizar el mismo proceso.

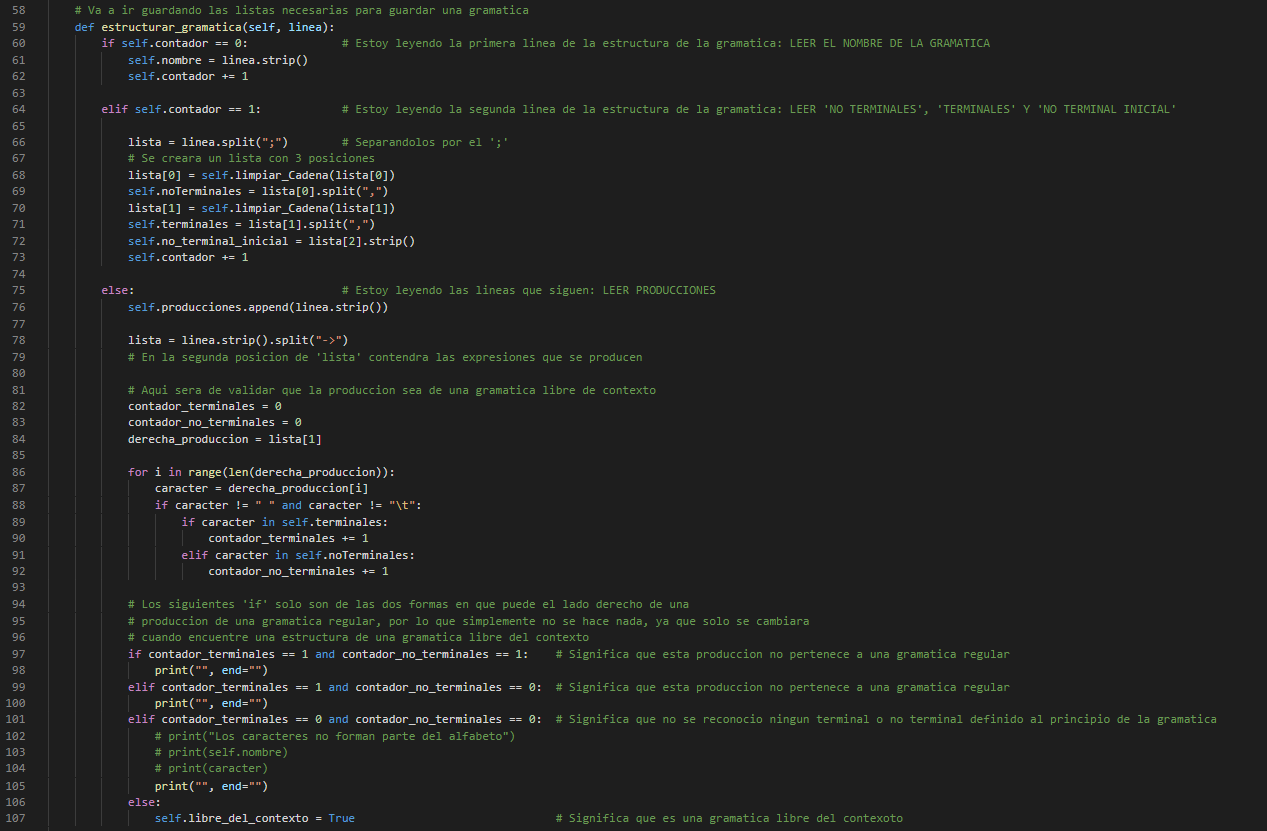


* **estructurar\_gramatica():** Esta función recibe por parámetro la línea del archivo en lectura en el base al valor almacenado en la variable *contador,* realiza una determinada tarea.

1. Si el contador es igual a 1, almacena en la variables *nombre* el nombre de la gramática, ya que es lo primero que está.
2. Si el contador es igual a 2, almacena la lista de terminales, no terminales y el no terminal inicial, ya que esta es la segunda parte de la gramática.
3. Si no es ni 1 ni 2, entonces significa que lo que va a estar leyendo son todas las producciones de la gramática.

Y este proceso continuo hasta encontrar un ‘\*’ en el archivo el cual indica el fin de la gramática, y el contador vuelve a 0.

Además, valida que la gramática sea libre del contexto, ya que la cantidad de terminales y de no terminales era 1 en todas las expresiones del lado derecho de la producción significa no es exclusivamente independiente del contexto, o si hubo como máximo un terminal y ningún terminal en la expresión del lado derecho de la producción o si no hubo ningún terminal o no terminal. Todos los casos anteriores hacen que esa gramática no se cargue al sistema.



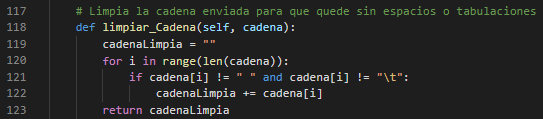
* **getNombres\_Gramaticas():** Retorna el nombre de todas las gramáticas cargadas en el sistema.



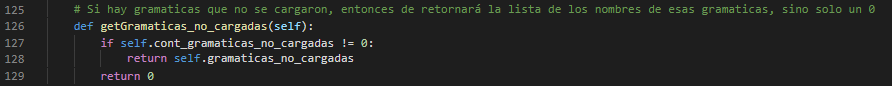
* **obtener\_objeto\_gramatica():** Recibe el nombre de la gramática que se desea encontrar y retorna el objeto gramática.



* **limpiar\_Cadena():** Recibe por parámetro una cadena y lo que realiza es eliminar los espacios en blanco o tabulaciones de la cadena, retornando así, una cadena en donde todo este ‘junto’, sin espacios.



* **getGramaticas\_no\_cargadas():** Retorna el nombre de todas las gramáticas no cargadas en el sistema.



* **Automata\_de\_Pila**

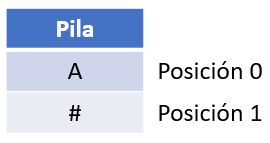
En este módulo se implementa un autómata de pila capaz de reconocer cadenas que se derivan de las gramáticas más comunes.

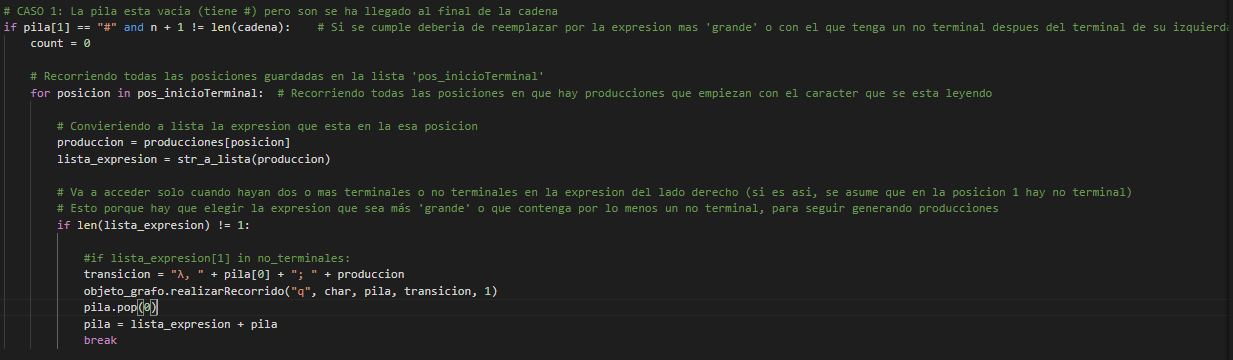
* **analizar\_Cadena():** Recibe por parámetro un objeto de la clase *Graphviz* (*obj\_grafo*)*,* la cadena a reconocer, las transiciones, los terminales, no terminales y el no terminal inicial. Esta función trabaja en base a estados, en donde los estas ‘i’ y ‘p’ nunca cambian, mientras que es en el estado ‘q’ en donde dependiendo del carácter en lectura y lo que esta en la cima de la pila, el automata actuara de forma distinta.

Para este autómata se consideraron los siguientes casos:

1. Caso 1: Si en la cima de la pila (posición 0) hay no terminal y en la posición 1 está el símbolo ‘#’ y aun no se termina de leer la cadena, entonces, se busca una producción que en el lado derecho tenga un dos o más terminales y no terminales que empiecen con el carácter que estoy leyendo, en este caso ‘a’.

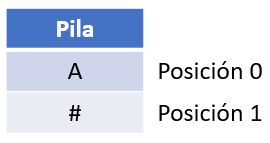
Cadena en lectura: aabz

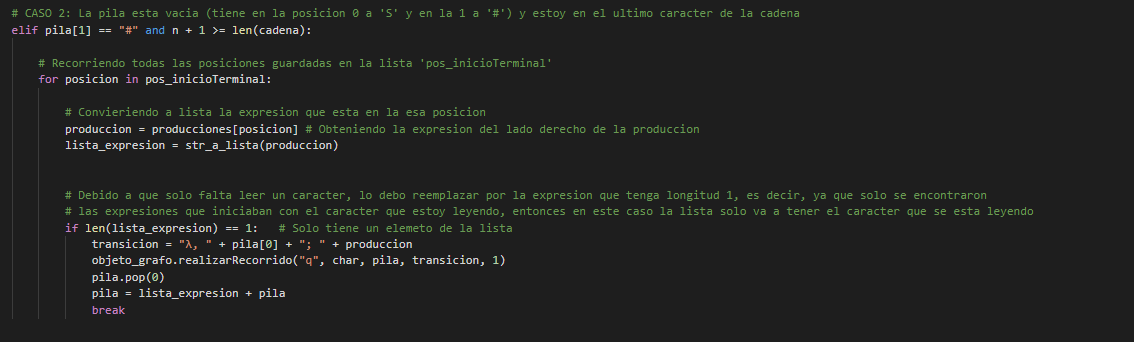




1. Caso 2: Si en la cima de la pila hay un no terminal y en la posición 1 esta el símbolo ‘#’ y la cadena en lectura es ‘a’, entonces que busque producciones del no terminal de la cima que solo produzcan el carácter en lectura, en este caso la letra ‘a’.

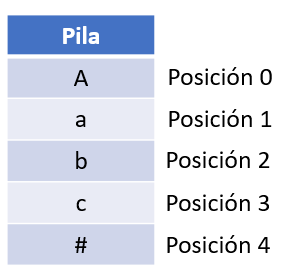
Cadena en lectura: a

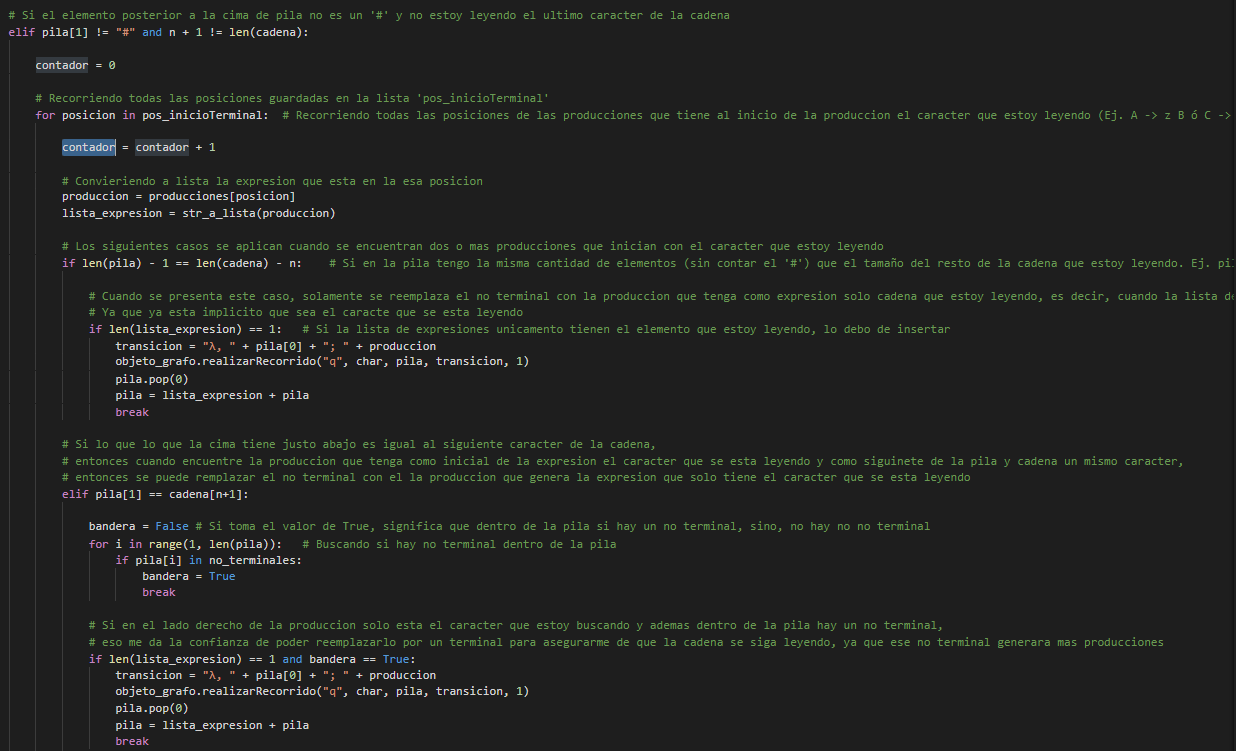




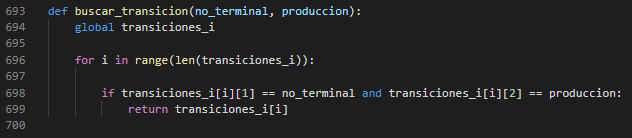
1. Caso 3: Si hay un no terminal en la cima de la pila, hay más contenido en la pila, pero la cadena aún no se lee por completo, entoces, hay que sustituir ese no terminal por una producción que genere una producción que tenga solo que carácter que se está leyendo.

Cadena: aabz

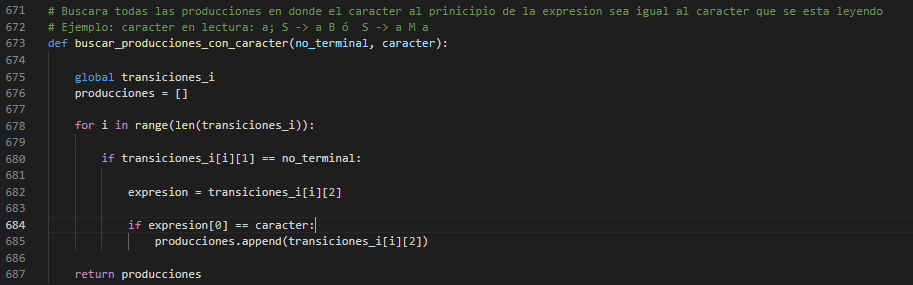




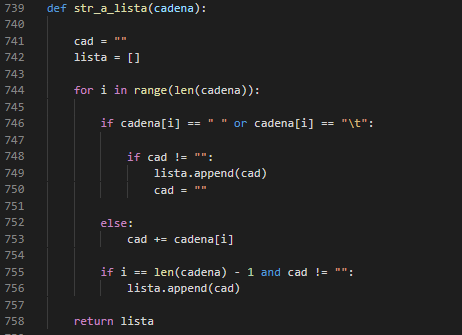
* **buscar\_transicion():** Recibe por parámetro el no terminal y la producción del cual se quiere encontrar sus transiciones, retornado la transición encontrada.



* **buscar\_producciones\_con\_no\_terminal():** Recibe por parámetros el no terminal del cual se desea buscar sus producciones y el carácter con el que debe empezar esa producción. Retorna todas las producciones que empiecen con ese carácter.



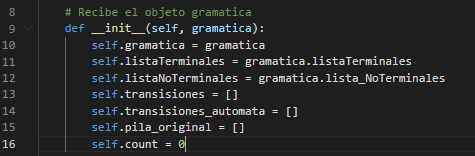
* **str\_a\_lista():** Recibe la cadena que se quiere convertir a lista.



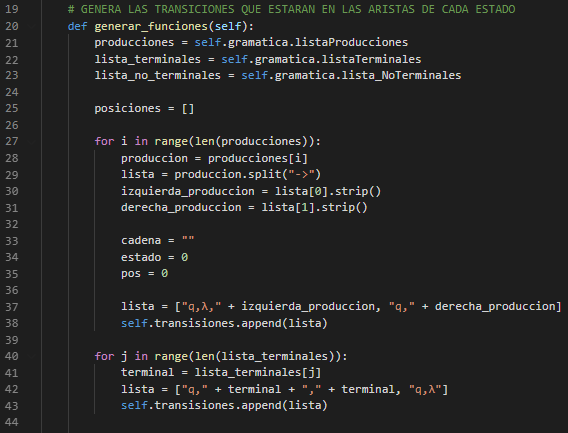
* **Graphviz.py**

Realiza toda la estructura para que pueda ser representado en un grafo todas las iteraciones.

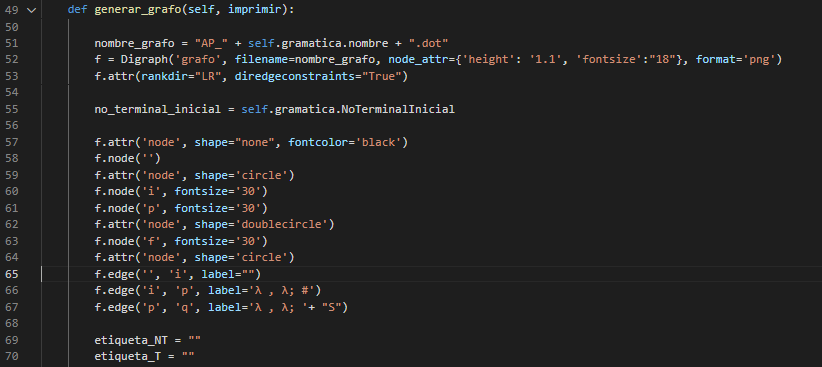
* **\_\_init\_\_():** Recibe por parámetro el objeto gramática y obtiene de este toda la información necesaria para generar el grafo.



* **generar\_funciones():** Genera la estructura de todas las transiciones de la forma: (q,$,A; q,aAa).

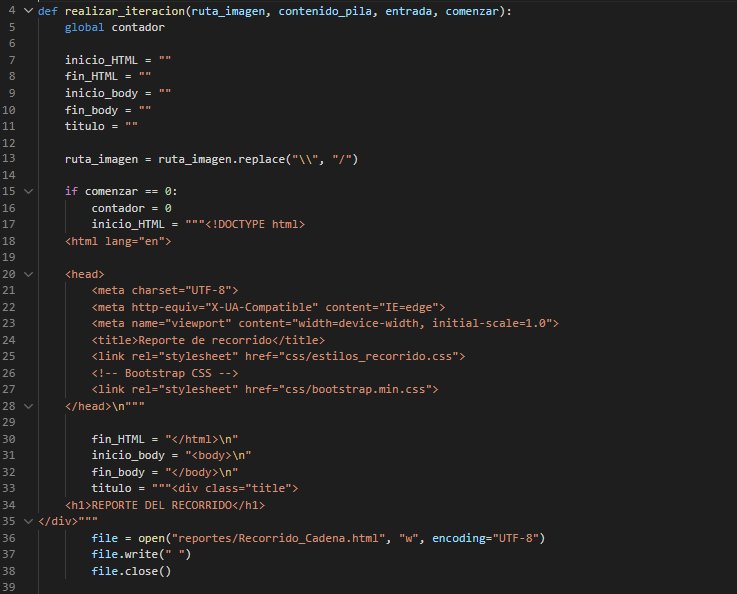


* **generar\_grafo():** Recibe por parámetro una variable por nombre ‘imprimir’, la cual, al tener el valor de True, generará el grafo y retornara la imagen para se colocada en el reporte html, si es False, únicamente genera la imagen pero no retorna nada.



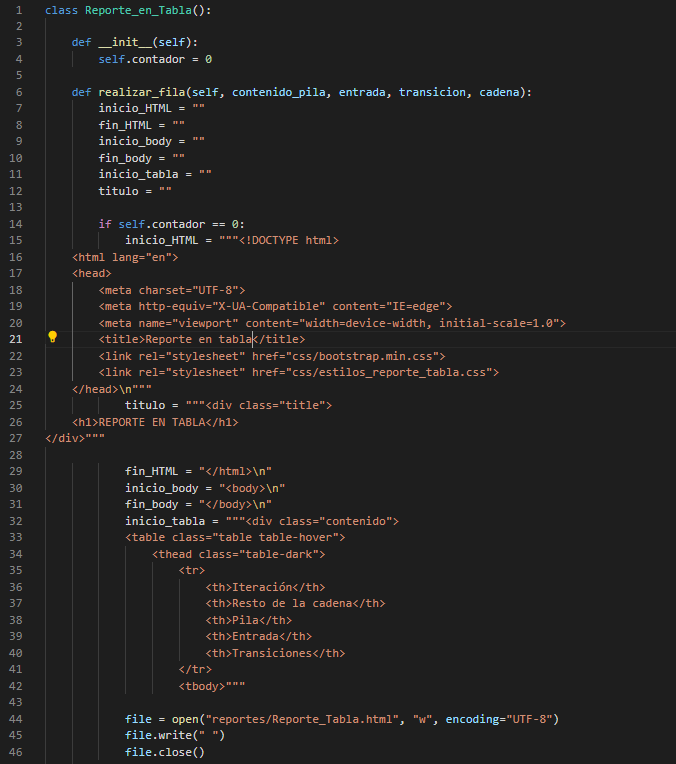
* **Recorrido\_Cadena.py**

Genera la estructura de html en donde se mostrará todo el recorrido para validar una cadena ingresada por el usuario.



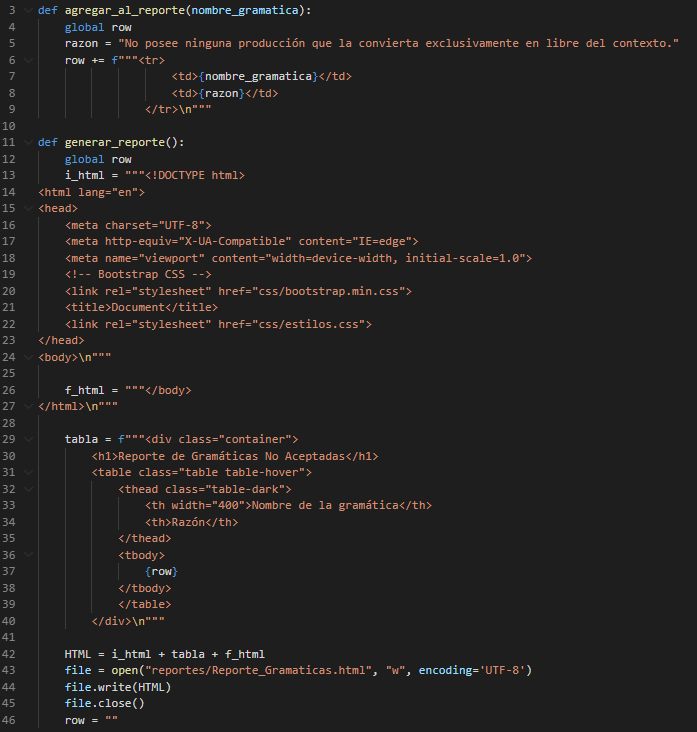
* **Reporte\_Tabla.py**

Genera la estructura de html en donde se mostrará todo el recorrido para validar una cadena ingresada por el usuario, pero esta vez representado en una tabla.

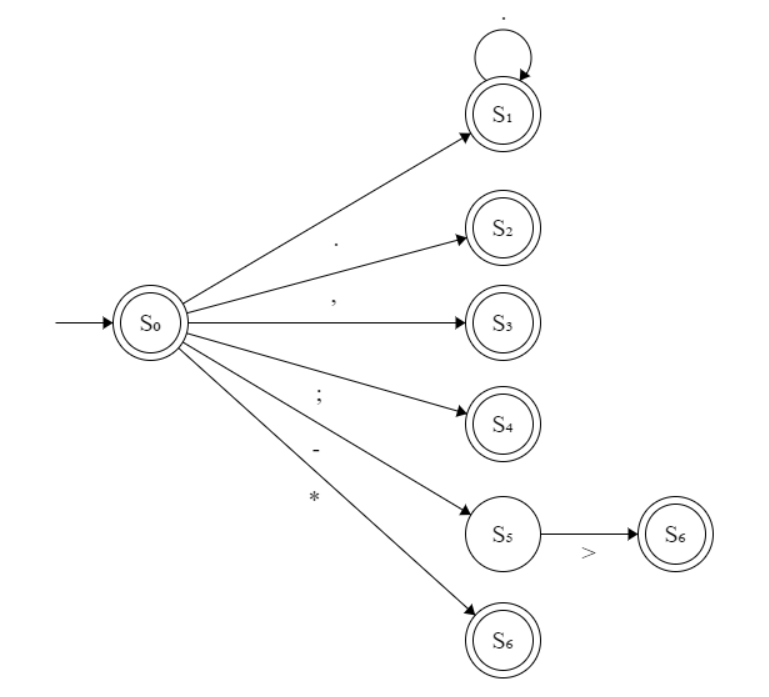


* **Reporte\_Gramaticas.py**

Genera la estructura de html en donde se mostrará una tabla de todas las gramáticas que no fueron cargadas en el sistema al no ser independientes del contexto.



# AUTÓMATA FINITO DETERMINISTA



El autómata puede recibir un carácter ‘n’ número de veces y este sería aceptado, asi como también un ‘,’, ‘;’, ‘\*’ y una secuencia que forme ‘->’, los cuales estarán es un estado de aceptación, con esto se valida que el archivo tenga esto y únicamente esto para la lectura del archivo.