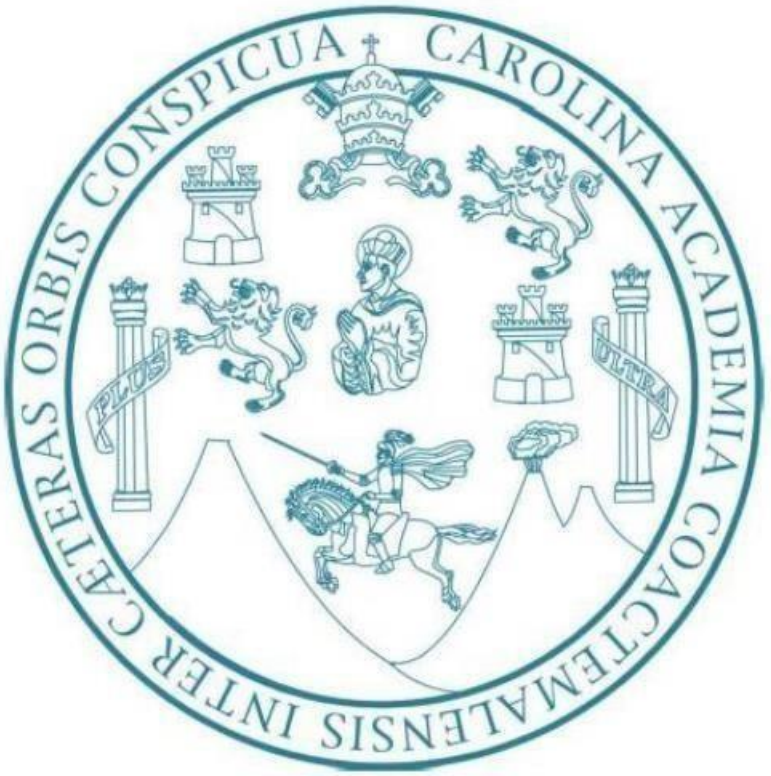
# Universidad de San Carlos de Guatemala

Facultad de Ingeniería
Lenguajes Formales y de Programación
Sección: "B" Cat. Ing. Manuel Castillo
Tutor académico: Huriel Gómez

PROYECTO 1: Manual Técnico




Bryan Estiveen Alarcón Aldana
Carnet: 201800526

*ANALIZADOR LÉXICO - PROYECTO*
*Desarrollo de aplicación de escritorio con enfoque en él análisis léxico y sintáctico.*

# Introducción

Este proyecto tiene como objetivo conseguir que los estudiantes de la facultad de ingeniería en ciencias y sistemas, los cuales son nuevos desarrolladores de software en la tecnología de Python, estén listos en el movimiento con la programación regida bajo las expresiones regulares y el desarrollo de aplicaciones del mundo real usando el lenguaje y la plataforma de Python.

La tecnología de Python se utiliza para desarrollar aplicaciones para un amplio alcance de entornos, desde dispositivos del consumidor hasta sistemas empresariales heterogéneos. Por ende, es importante introducir a cada estudiante en este lenguaje de programación, y con este proyecto se pretende realizar lo anteriormente descrito. En función a esto, el proyecto nos invita a investigar las distintas expresiones regulares y como implementarlas sin la utilización de librerías, con el fin de elaborar las instrucciones propuestas en el instructivo y así llevar a cabo dicha práctica.

Como cualquier lenguaje de programación, el lenguaje utilizado en Python tiene su propia estructura, reglas de sintaxis y paradigma de programación.

# Diccionario de métodos usados

- Main: El método o en este caso clase principal de Python es el punto de entrada de cualquier programa en python, en el cual se encuentra el medio por el cual se ejecutan los métodos consiguientes del programa.

- AnalisisSintacticoJS: En este método, se contiene todas las instrucciones referentes al análisis sintáctico para una entrada tipo calculadora, ya que obtiene los tokens del análisis léxico de JS y verifica si están en el orden correcto para su validación.

- AnalisisLexicoJS: En este método, esta contenido las funciones para el análisis léxico de un archivo tipo JS, en el cual se comparan los tokens provenientes de la cadena de entrada con el archivo tokensJS, con el fin de verificar si están contenidos en el listado, de lo contrario se marcaría un error léxico. De igual manera, en esta clase están contenidos los estados por los que pasa cada token, para verificar que tipo de entrada es.

- TokenJS: En este método esta contenido un listado de caracteres permitidos en JS, de igual manera, se retorna el valor de cada uno para su posterior análisis léxico y sintáctico.

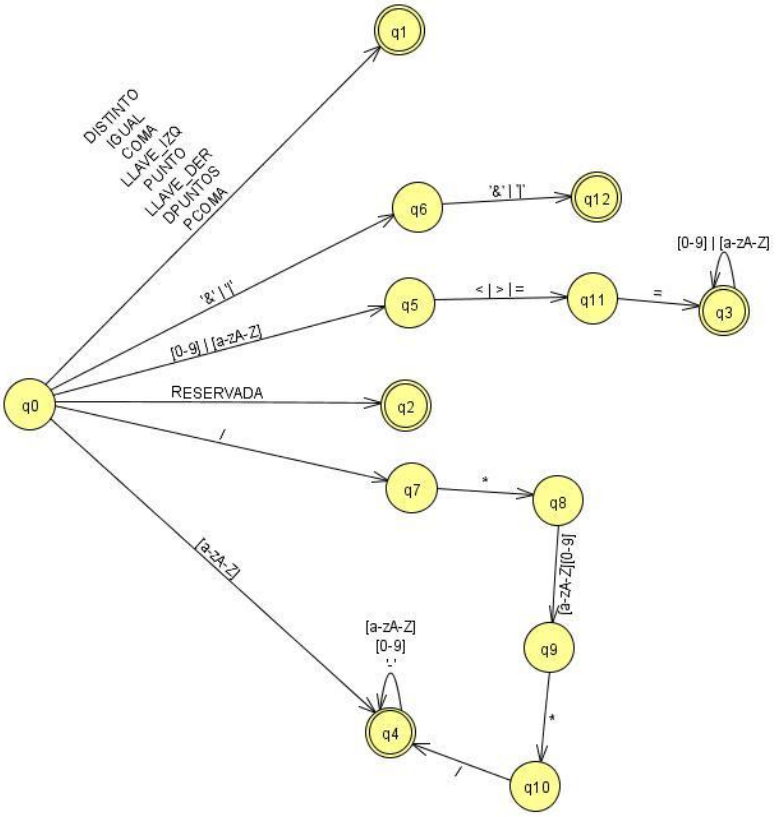
- AnalisisLexicoCSS: En este método, esta contenido las funciones para el análisis léxico de un archivo tipo CSS, en el cual se comparan los tokens provenientes de la cadena de entrada con el archivo tokensCSS, con el fin de verificar si están contenidos en el listado, de lo contrario se marcaría un error léxico. De igual manera, en esta clase están contenidos los estados por los que pasa cada token, para verificar que tipo de entrada es.

- TokenCSS: En este método esta contenido un listado de caracteres permitidos en CSS, de igual manera, se retorna el valor de cada uno para su posterior análisis léxico.

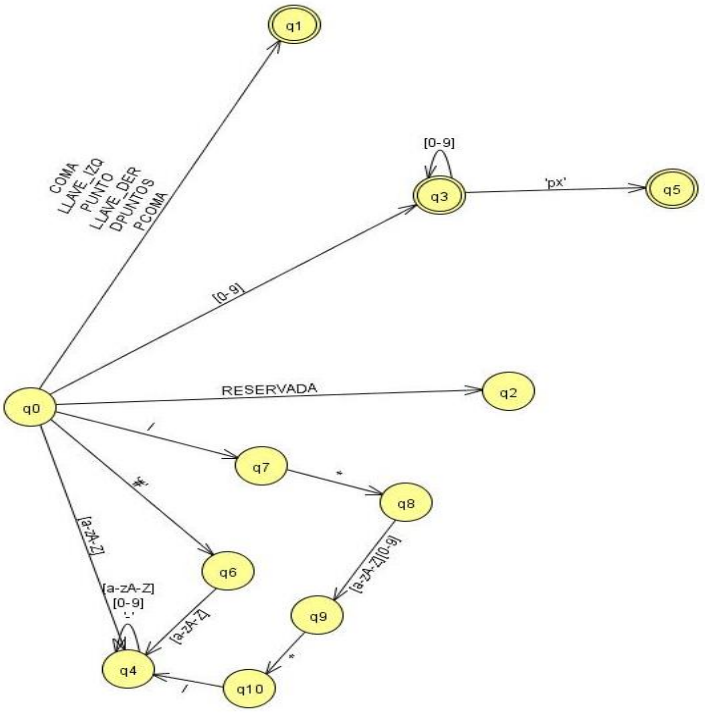
- AnalisisLexicoHTML: En este método, esta contenido las funciones para el análisis léxico de un archivo tipo HTML, en el cual se comparan los tokens provenientes de la cadena de entrada con el archivo tokensHTML, con el fin de verificar si están contenidos en el listado, de lo contrario se marcaría un error léxico. De igual manera, en esta clase están contenidos los estados por los que pasa cada token, para verificar que tipo de entrada es.

- TokenHTML: En este método esta contenido un listado de caracteres permitidos en HTML, de igual manera, se retorna el valor de cada uno para su posterior análisis léxico.

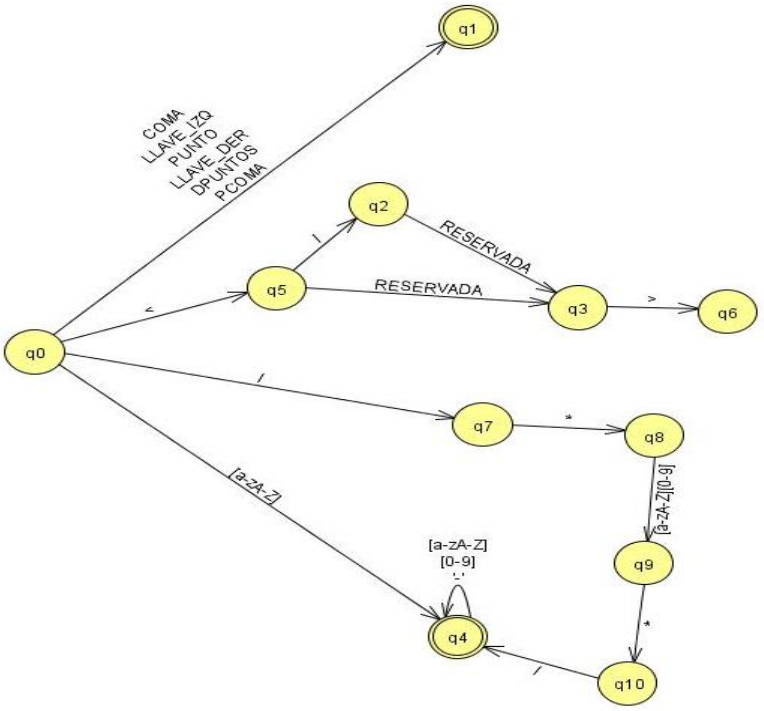
AutomataJS:




AutomataCSS:



AutomataHTML:




**Expresiones regulares:**

LETRA: [a-zA-Z]
Numero: [0-9]

//Expresiones regulares para valores a asignar o algún id
ID -> ('#')? LETRA (LETRA |NUMERO|'-')*
NUMERO -> (NUMERO)+ ('LETRA')?

//palabras reservadas //CSS
color : 'color'
fsize : 'font-size'
pos : 'position'
top : 'top'
back_color : 'background-color'
marginTop : 'margin-top'

//palabras reservadas //JS
if : 'if'
else : 'else'
for : 'for'
break : 'break'
switch : 'switch'

//palabras reservadas //HTML
html : 'html'
border : 'border'
color : 'color'
body_color : 'body_color'
p : 'p'
h# : 'h#'

//Simbolos del lenguaje
'{' : LLAVE_IZQ
'}' : LLAVE_DER
':' : DPUNTOS
';' : PCOMA
',' : COMA
'!' : DISTINTO
'&' : CONJUNCION
'|' : DISYUNCION
'+' : MAS
'-' : MENOS
'*' : ASTERISCO
'/' : BARRA
'<' : MENOR
'>' : MAYOR
'"' : COMILLAS
'=' : IGUAL
'(' : PAREN_IZQ
')' : PAREN_DER

//Algunos ejemplos de los tokens

| [Token] | [Informal Description] | [Sample Lexemes] |
|---|---|---|
| if | characters i, f | if |
| else | characters e, l, s, e | else |
| for | characters f, o, r | for |
| break | characters b, r, e, a, k | break |
| switch | characters s, w, i, t, c, h | switch |
| html | characters h, t, m, l | html |
| border | characters b, o, r, d, e, r | border |
| color | characters c, o, l, o, r | color |
| back_color | characters b, a, c, k, c, o, l, o, r | back_color |
| comparison | < or > or <= or >= or == or != | <=, != |
| ID | letter followed by letters and digits | pi, score, D2 |
| ENTERO | any numeric constant | 3.14159, 0, 6.02e23 |
| LITERAL | anything but ", surrounded by "'s | "core dumped" |

## Gramatica:

BEGIN -> ID '(' LISTA_REGLA ')'
| ENTERO
| DECIMAL
LISTA_REGLA -> REGLA '+|-|*|(|)|/' LISTA_REGLA
| EPSILON
REGLA -> ID '+|-|*|(|)|/' VALOR
VALOR -> ENTERO
| DECIMAL