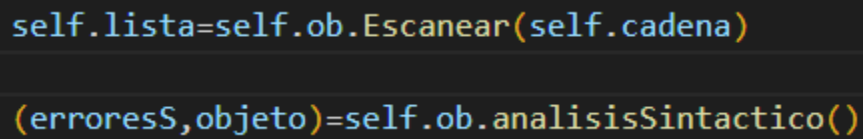


# Manual técnico

El proyecto se realizó en el lenguaje de programación Python utilizando el paradigma de programación orientada a objetos (POO), así mismo se utilizó la librería Tkinter para generar la interfaz gráfica.

El proyecto consta de un analizador léxico y sintáctico que corrobora la sintaxis del lenguaje previsto.

El analizador léxico genera una lista de tokens que luego verifica el analizador sintáctico.



```
self.lista=self.ob.Escanear(self.cadena)
(erroresS,objeto)=self.ob.analisisSintactico()
```

Lista de  
tokens

## Estructura del analizador léxico:

El analizador léxico consta de 7 estados enumerados del 0 al 6 los cuales están conformados de la siguiente manera:

### Estado 0:

El estado 0 comprueba los símbolos válidos del lenguaje tales como el símbolo <, (,), etc. Así mismo los saltos de línea.

### Estado 1:

Detecta los números enteros y pasa al siguiente estado si viene un punto decimal.

### Estado 2:

El estado 2 funciona de paso para los siguientes números después del punto decimal.

### Estado 3:

Termina de verificar los números después del número decimal.

### Estado 4:

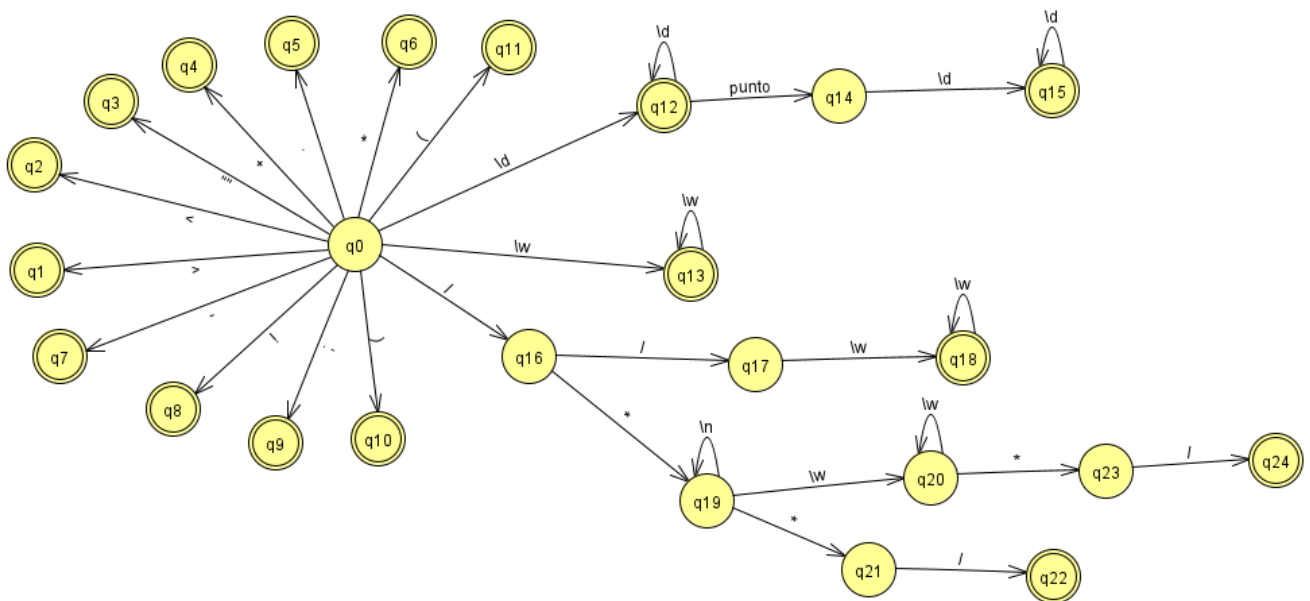
Determina los comentarios del tipo `'''` y todo el contenido que este contenga.

### Estado 5:

Verifica las cadenas que provengan del archivo de entrada, así como las palabras reservadas que el programa acepta.

### Estado 6:

Este estado 6 permite verificar y controlar los comentarios multilínea, así como todo el contenido que este contenga.



## Estructura del analizador sintáctico:

El analizador sintáctico se compone de 3 gramáticas diferentes para la estructura del archivo de entrada, 1 gramática por cada estructura del lenguaje.

### Estructura 1:

Esta es la estructura donde se definen los controles.

```

<!--Controles
    Contenedor contlogin;
    Contenedor cont;
    Texto in1;
    Texto in2;
    Boton boton;
    Clave clv;
    Etiqueta et;
Controles-->

```

Esta gramática de manera simple se compone de lo siguiente:

<,!,-, Controles, elemento, identificador, puntoycoma ,Controles ,-,>

Definido en la lógica de la programación de forma equivalente.

```

self.Parseo("ANG_A", "<")
self.Parseo("EXCLAM", "!")
self.Parseo("MENOS", "-")
self.Parseo("MENOS", "-")
self.Parseo("tControles", "Controles")
self.gramatica_contenido()
self.Parseo("tControles", "Controles")
self.Parseo("MENOS", "-")
self.Parseo("MENOS", "-")
self.Parseo("ANG_C", ">")

```

Entonces, la gramática quedaría representada como:

ANG\_A EXCLAM MENOS MENOS CONTROLES gramatica\_contenido  
CONTROLES MENOS MENOS ANG\_C

gramatica\_contenido → gramatica\_contenido elementos

elementos → elemento tkID PUNTCOMP

donde el terminal elementos representa el siguiente código

```

self.contiene(self.Salida[self.cont].getTipo(), "un elemento")
self.Parseo("tkID", "un identificador")
self.Parseo("PUNTCOM", ";")

```

Dentro de una función recursiva se encuentra este conjunto de instrucciones que se ejecuta siempre y cuando se encuentre dicha gramática dentro de la estructura controles.

Estructura 2:

De la misma manera que la estructura 1, la estructura de propiedades cumple con la siguiente gramática

ANG\_A EXCLAM MENOS MENOS PROPIEDADES gramática\_prop  
PROPIEDADES MENOS MENOS ANG\_C

```
self.Parseo("ANG_A", "<")
self.Parseo("EXCLAM", "!")
self.Parseo("MENOS", "-")
self.Parseo("MENOS", "-")
self.Parseo("tProp", "propiedades")
self.gramaticaprop()
self.Parseo("tProp", "propiedades")
self.Parseo("MENOS", "-")
self.Parseo("MENOS", "-")
self.Parseo("ANG_C", ">")
```

*Representación de la gramática en código Python*

gramaticaprop → gramaticaprop propiedades

propiedades → tkID PUNTO propiedad PARE\_A params PARE\_C PUNTCOM

params → tkID

| NUM

| Cadena

La función Parseo verifica si realmente viene el token especificado en la gramática.