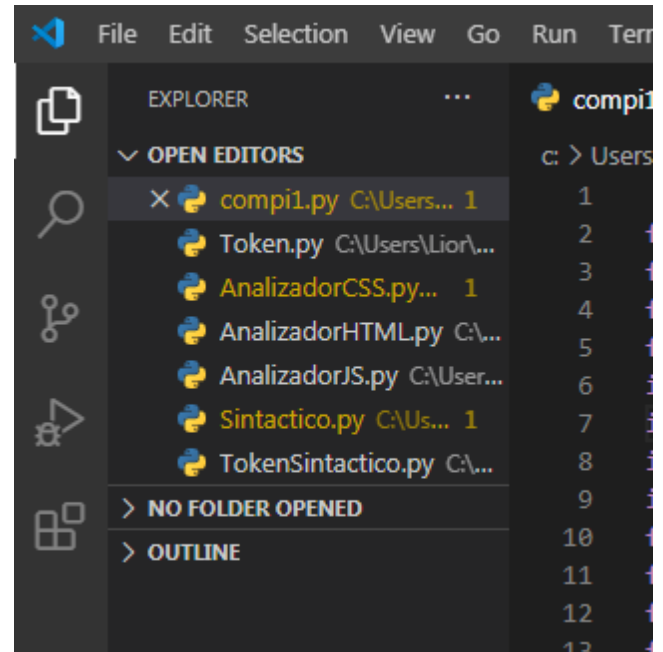


Manual técnico

El programa cuenta con 7 clases:

1. **Compi1.py:** clase encargada de la interfaz gráfica.
2. **Token.py:** clase encargada de guardar los tokens recopilados en el análisis.
3. **AnalizadorCSS.py:** clase encargada de generar el análisis del lenguaje CSS.
4. **AnalizadorHTML.py:** clase encargada de generar el análisis del lenguaje HTML.
5. **AnalizadorJS.py:** clase encargada de generar el análisis del lenguaje JS.
6. **Sintactico.py:** clase encargada de generar el análisis sintáctico de las operaciones aritméticas del lenguaje JS.
7. **TokenSintactico.py:** clase encargada de guardar los tokens recopilados en el análisis JS.



Cada clase recibe como sobrecarga el párrafo de entrada, dicho párrafo lo separa por letras y lo recorre con un while, aumentando su posición y moviéndose entre estados para reconocer los tokens del lenguaje que se este ejecutando.

Recorrido del párrafo con un while

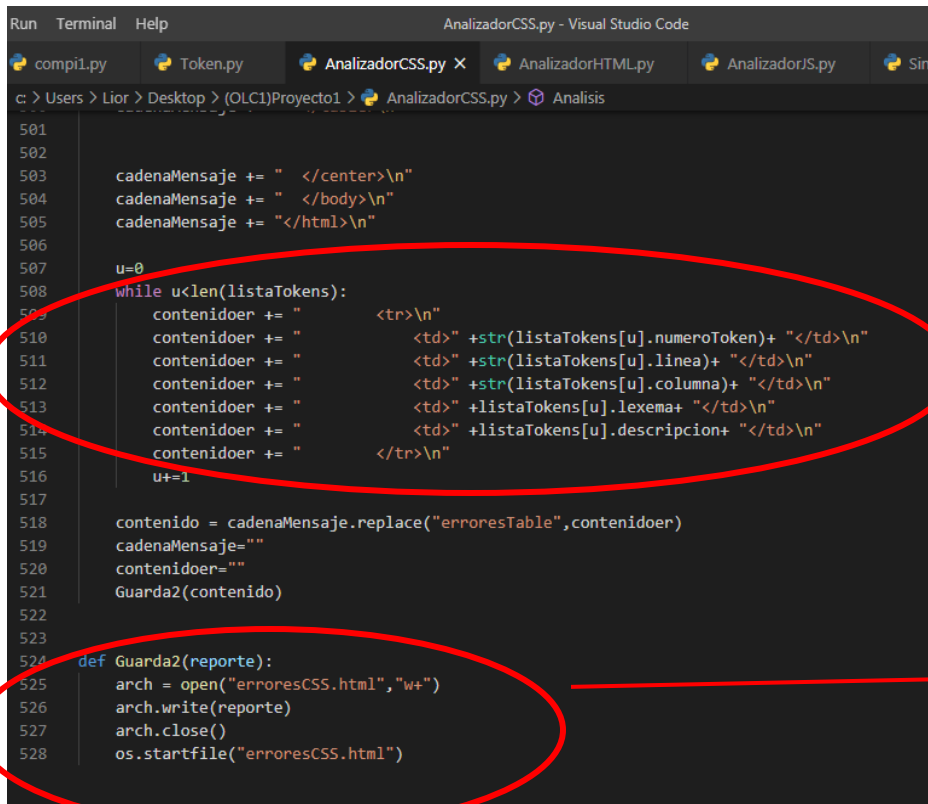
```
Run Terminal Help
AnalizadorHTML.py - Visual Studio Code
compil.py Token.py AnalizadorCSS.py AnalizadorHTML.py X AnalizadorHTML.py
c: > Users > Lior > Desktop > (OLC1)Proyecto1 > AnalizadorHTML.py > codigoERRORES
1 import string
2 from Token import ClaseToken
3 import os
4
5 # ADRIANA GÓMEZ
6 # 201504236
7 # COMPILADORES 1
8
9
10
11 listaError = list()
12
13 def AnalizaHTML(parrafo):
14     estado = 0
15     numerotk = 1
16     palabra = ""
17     control = 0
18     columna = 0
19     fila = 0
20     listaError.clear()
21     #---- RECORRIENDO EL PARRAFO ----#
22     while control < len(parrafo):
23
24
25         if estado == 0:
26             if parrafo[control] == "\n":
27                 control+=1
28                 fila += 1
29                 estado = 0
30             elif parrafo[control] == " " or parrafo[control] == "\t":
31                 control += 1
32                 columna += 1
33                 estado = 0
34             elif parrafo[control] == "<":
35                 control+=1
36                 columna += 1
```

Movimiento entre estados

```
AnalizadorCSS.py - Visual Studio Code
py Token.py AnalizadorCSS.py X AnalizadorHTML.py AnalizadorJS.py
> Lior > Desktop > (OLC1)Proyecto1 > AnalizadorCSS.py > Analisis
if estado == 1:
    if parrafo[control] == "*":
        control+=1
        estado = 6
        numerotk += 1
        columna += 1
        palabra = "*"
        palbrabitacora+="\nS1 -> S6: token "+palabra
    else:
        #----- ERROR LEXICO ----#
        palabra = parrafo[control]
        control+=1
        estado = 6
        numerotk2 += 1
        columna += 1
        nuevo=ClaseToken(numerotk2,fila,columna,palabra,"error lexico")
        listaTokens.append(nuevo)

if estado == 2:
    if parrafo[control] == "{":
        control += 1
        estado = 9
        numerotk += 1
        columna += 1
        palabra = "{"
        palbrabitacora+="\nS2 -> S9: token "+palabra
    else:
        #----- ERROR LEXICO ----#
        palabra = parrafo[control]
        control+=1
        estado = 9
```

Cada clase también genera su propia lista de errores dinámicamente, cada vez que se ejecuta el análisis, se genera su reporte respectivo de errores en HTML



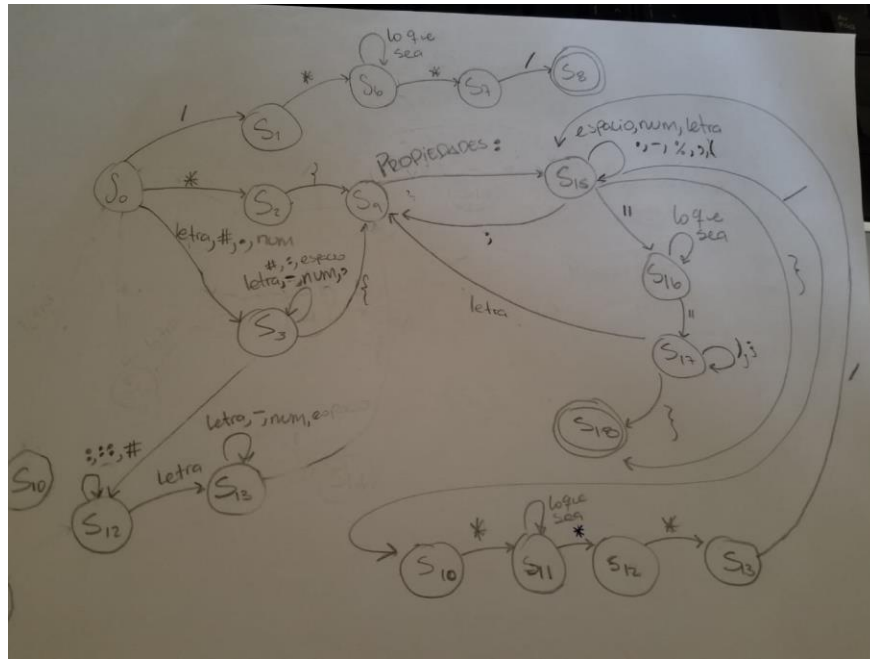
```
501
502
503 cadenaMensaje += " </center>\n"
504 cadenaMensaje += " </body>\n"
505 cadenaMensaje += "</html>\n"
506
507 u=0
508 while u<len(listaTokens):
509     contenidoer += " <tr>\n"
510     contenidoer += " <td>" +str(listaTokens[u].numeroToken)+ "</td>\n"
511     contenidoer += " <td>" +str(listaTokens[u].linea)+ "</td>\n"
512     contenidoer += " <td>" +str(listaTokens[u].columna)+ "</td>\n"
513     contenidoer += " <td>" +listaTokens[u].lexema+ "</td>\n"
514     contenidoer += " <td>" +listaTokens[u].descripcion+ "</td>\n"
515     contenidoer += " </tr>\n"
516     u+=1
517
518 contenido = cadenaMensaje.replace("erroresTable",contenidoer)
519 cadenaMensaje=""
520 contenidoer=""
521 Guarda2(contenido)
522
523
524 def Guarda2(reporte):
525     arch = open("erroresCSS.html","w+")
526     arch.write(reporte)
527     arch.close()
528     os.startfile("erroresCSS.html")
```

Recorre su lista de errores léxicos cada lenguaje

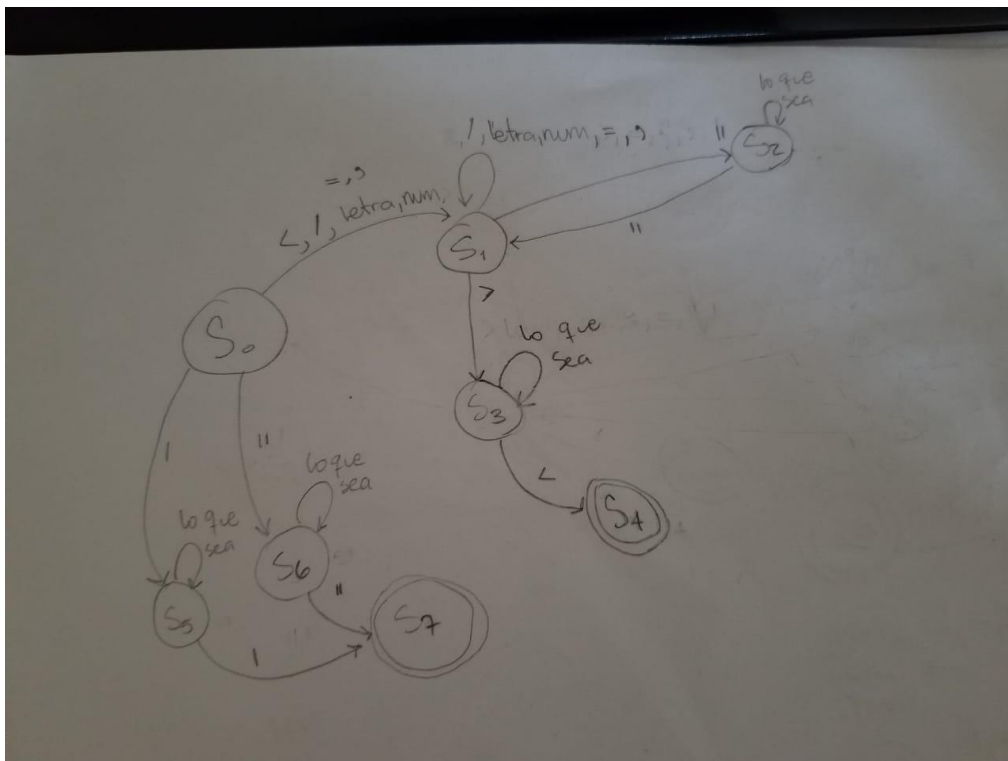
Cada clase crea una página HTML y manda su lista de errores.

Automatas implementados en el análisis

- Autómata del lenguaje CSS



- Autómata del lenguaje HTML



- [illegible]

-

Requerimientos mínimos del sistema

- **Sistemas operativos admitidos**

Visual Studio 2017 se instalará y ejecutará en los siguientes sistemas operativos:

Windows 10 versión 1507 o posteriores: Home, Professional, Education y Enterprise (LTSC y S no se admiten)

Windows Server 2016: Standard y Datacenter

Windows 8.1 (con la actualización 2919355): Core, Professional y Enterprise

Windows Server 2012 R2 (con la actualización 2919355): Essentials, Standard y Datacenter

Windows 7 SP1 (con las actualizaciones más recientes de Windows): Home Premium, Professional, Enterprise y Ultimate

- **Hardware**

Procesador de 1,8 GHz o superior. Doble núcleo o superior recomendado.

2 GB de RAM; 4 GB de RAM recomendado (mínimo de 2,5 GB si se ejecuta en una máquina virtual)

Espacio en disco duro: hasta 130 GB de espacio disponible, en función de las características instaladas; las instalaciones típicas requieren entre 20 y 50 GB de espacio libre.

Velocidad del disco duro: para mejorar el rendimiento, instale Windows y Visual Studio en una unidad de estado sólido (SSD).

Tarjeta de vídeo que admita una resolución de pantalla mínima de 720p (1280 x 720); Visual Studio funcionará mejor con una resolución de WXGA (1366 x 768) o superior.

- **Idiomas compatibles**

Visual Studio está disponible en alemán, checo, chino (simplificado), chino (tradicional), coreano, español, francés, inglés, italiano, japonés, polaco, portugués (Brasil), ruso y turco. Puede seleccionar el idioma de Visual Studio durante la instalación. El instalador de Visual Studio está disponible en los mismos catorce idiomas y coincide con el idioma de Windows, si está disponible.

Nota: La integración de Office para Team Foundation Server de Visual Studio 2017 está disponible en los diez idiomas compatibles con Visual Studio Team Foundation Server 2017.

- **Requisitos adicionales**

1. Se necesitan derechos de administrador para instalar Visual Studio.
2. Se requiere .NET framework 4.5.2 o una versión superior para instalar Visual Studio. Visual Studio requiere .NET Framework 4.7.2 para ejecutarse, pero se instalará durante la configuración.
3. La edición LTSC de Windows 10 Enterprise y Windows 10 S no se admiten para el desarrollo. Para compilar aplicaciones que se ejecuten en Windows 10 LTSC y Windows 10 S, puede usar Visual Studio 2017.
4. Internet Explorer 11 o Microsoft Edge son necesarios para los escenarios relacionados con Internet. Algunas características podrían no funcionar a menos que se instalen estas soluciones o versiones posteriores.
5. Visual Studio no se admite en entornos virtualizados como Microsoft App-V para Windows o tecnologías de virtualización de aplicaciones de terceros.

6. Las opciones Interfaz mínima de servidor y Server Core no se admiten al ejecutar Windows Server.
7. Los contenedores de Windows no son compatibles, excepto con Visual Studio Build Tools 2017.
8. Para la compatibilidad con el emulador, se requieren las ediciones de Windows 8.1 Pro o Enterprise (x64). También es necesario un procesador que admita la Traducción de direcciones de segundo nivel (SLAT).
9. El desarrollo de aplicaciones Windows universales, incluidos el diseño, la edición y la depuración, requiere Windows 10. Windows Server 2016 y Windows Server 2012 R2 pueden utilizarse para crear aplicaciones Windows universales desde la línea de comandos.
10. Team Foundation Server 2017 Office Integration requiere Office 2016, Office 2013 u Office 2010.
11. Xamarin.Android requiere una edición de 64 bits de Windows y Java Development Kit (JDK) de 64 bits.
12. Se requiere PowerShell 3.0 o una versión posterior en Windows 7 SP1 para instalar el desarrollo móvil con cargas de trabajo de C++, JavaScript o .NET.