Manual Técnico

Datos de desarrollador

Lenguajes Formales y de Programación

Alvaro Norberto García Meza

Carné: 202109567

Detalles de desarrollo:

Se utilizo el lenguaje de programación Python y su herramienta integrada de desarrollo de interfaces gráficas, Tkinter, donde se realizó un analizador léxico de un archivo ".txt" con estructura HTML.

Descripción general:

Se solicita la lectura de código fuente, creando un programa el cual sea capaz de identificar un lenguaje dado, identificando los errores léxicos y ejecutando las instrucciones correspondientes.

Se listaron una serie de instrucciones las cuales deben de ser ejecutadas, cumpliendo con el formato asignado, generando un archivo HTML como salida.

De igual manera, se manejó errores los cuales se mostraron en formato de una tabla en un archivo HTML.

Paradigma utilizado:

Se implementaron dos paradigmas de programación el primero fue programación orientada a objetos donde se implemento dos clases una que hederá de la clase "Tk" para poder crear interfaces gráficas, dentro de esta clase se encuentra todas las herramientas que esta Liberia utiliza separada por módulos donde cada modulo representa una ventana anidada. Luego se utilizo una clase para el modelado de un curso donde este sea reutilizable durante la ejecución.

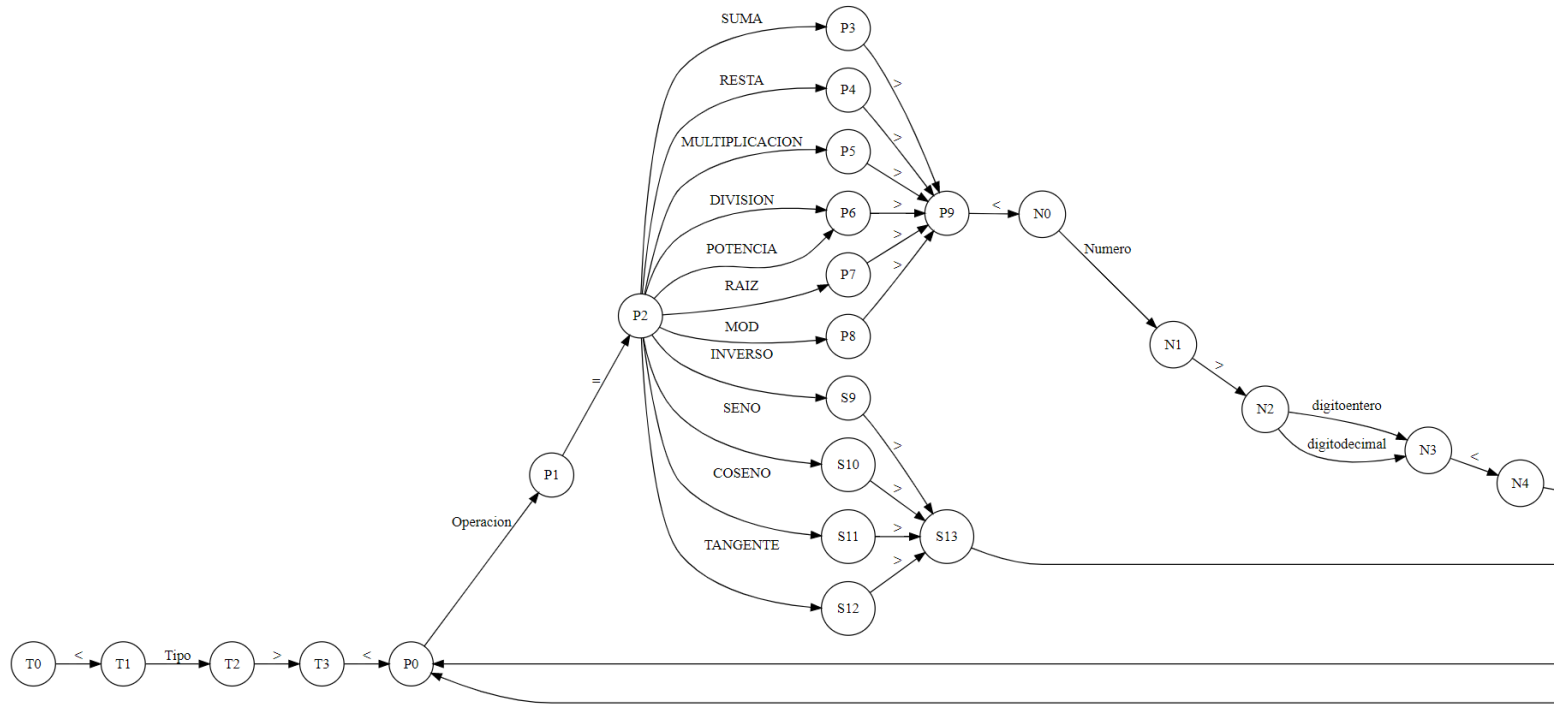
Por último, se trabajó con programación funcional para dividir las funciones de las ventanas fuera de su instancia como ventana misma.

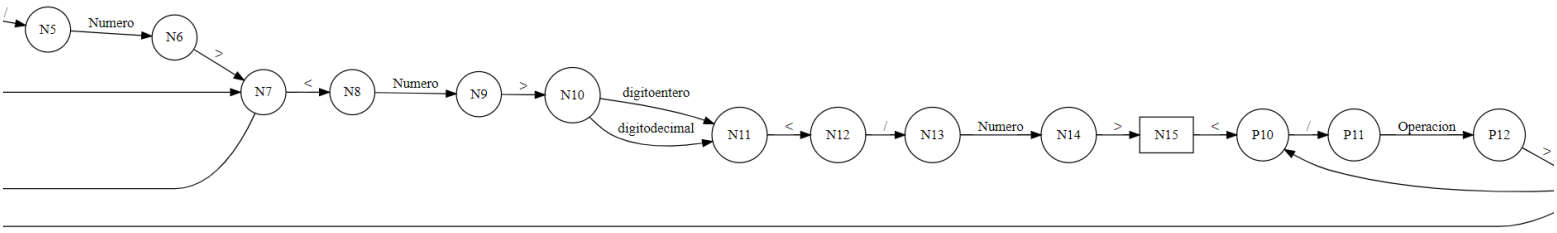
Diccionario de librerías:

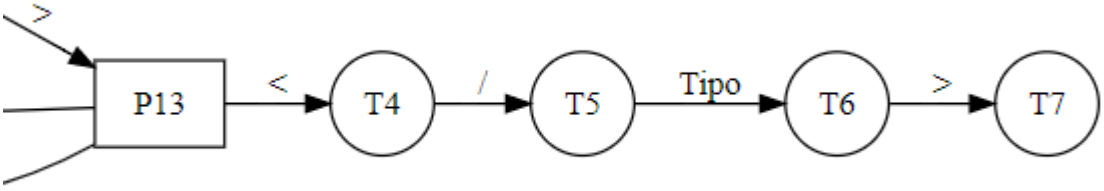
- **Tkinter**
  Es una librería integrada de Python la cual sirve para el desarrollo de interfaces gráficas. Proporciona un conjunto de herramientas e independientes para administrar ventanas.
  Widgets utilizados:
  - Clase tkinter como "tk.Tk" para inicializar un objeto como una aplicación.
  - Label: Para insertar texto estático en las ventanas.
  - Button: Botones para darle funcionalidad a nuestra aplicación al momento de darle clic.
  - Toplevel: Es un widget para crear ventanas hijas de la ventana principal.
  - Entry: Es un widget para ingresar valores de tipo texto.
  - StringVar: Es una variable especial de tkinter para almacenar valores de tipo String.

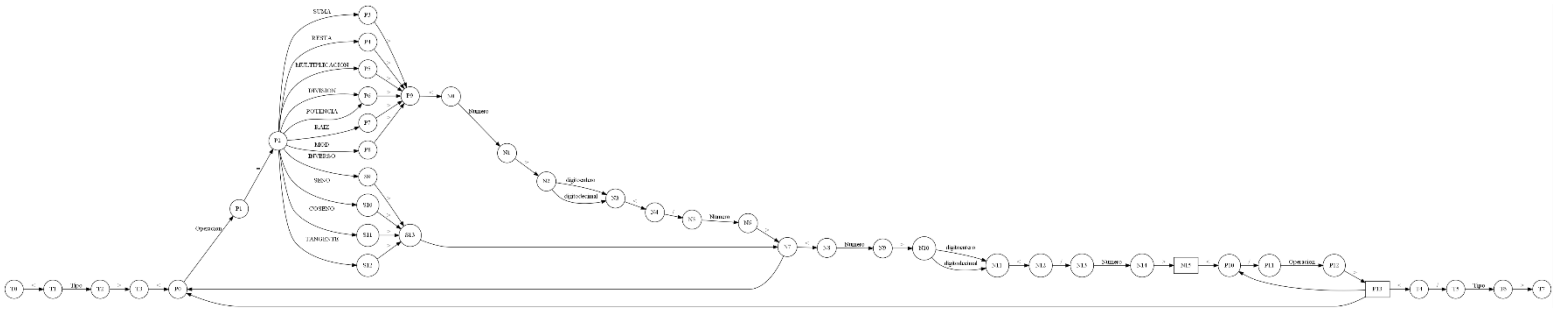
Autómata Finito Determinista (AFD)

AFD de etiqueta <Tipo> <Operacion> y <Numero> juntas

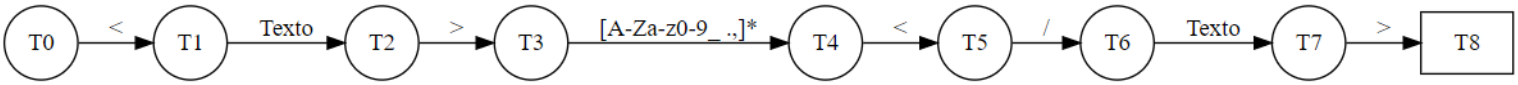




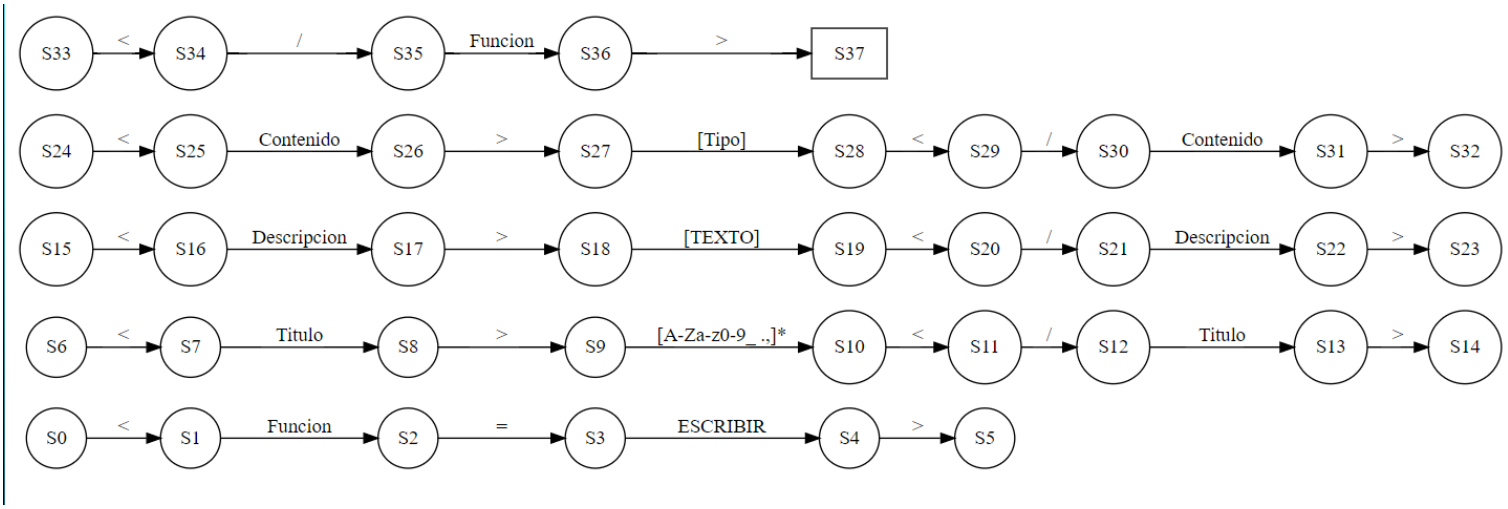




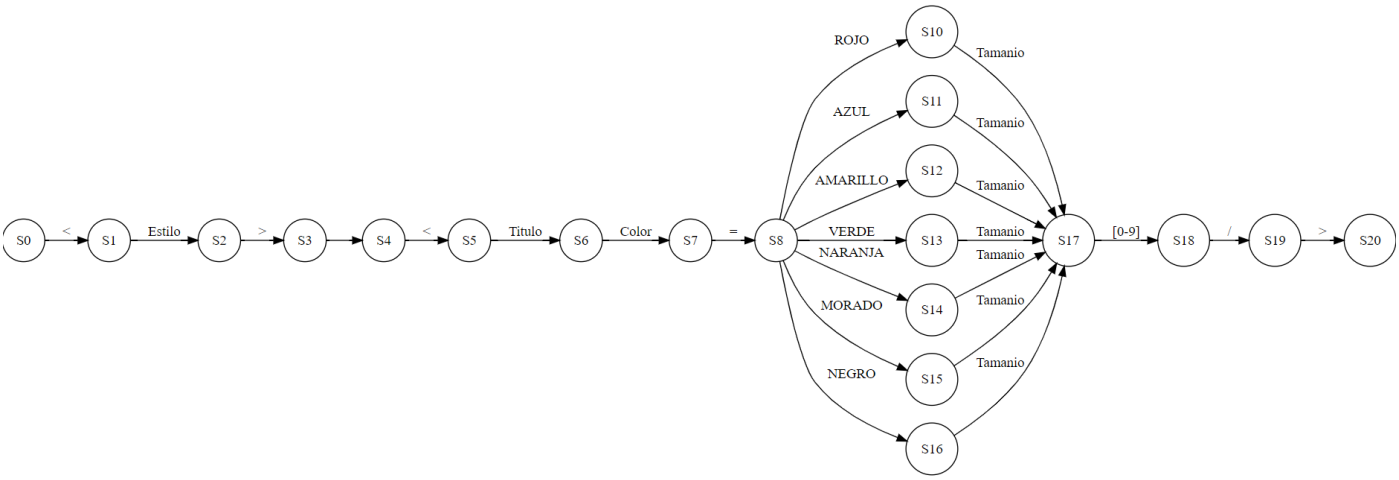
AFD de etiqueta <Texto>

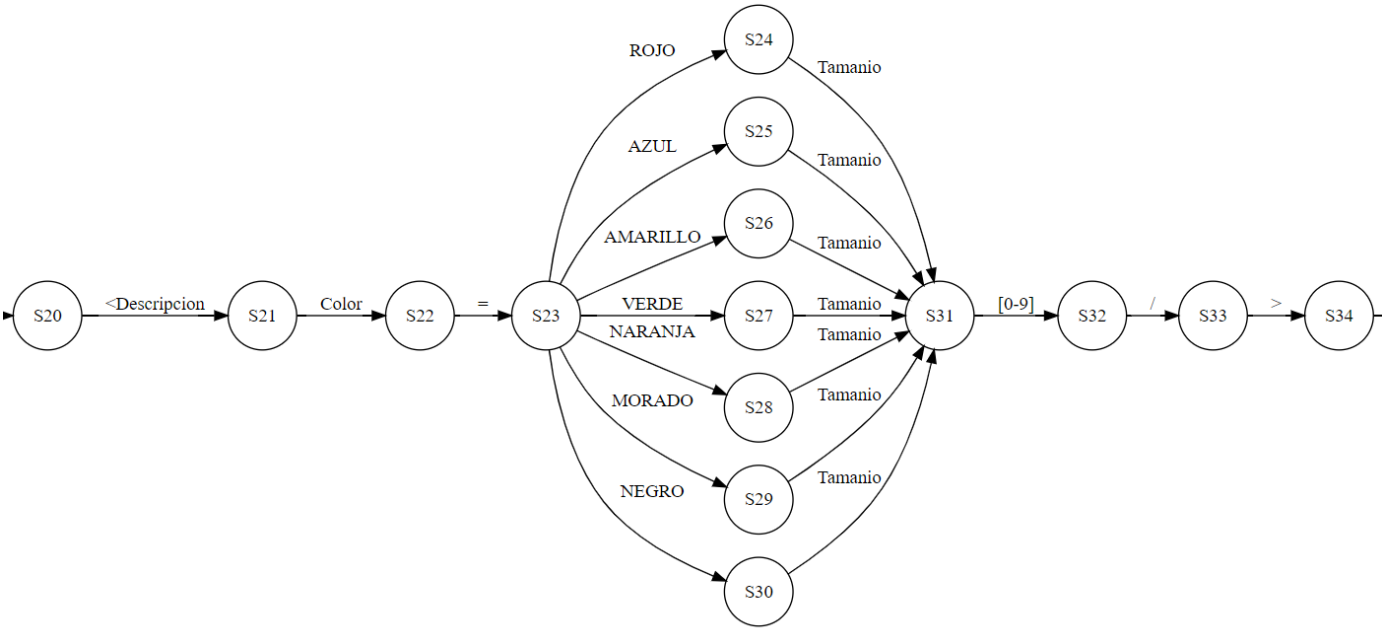


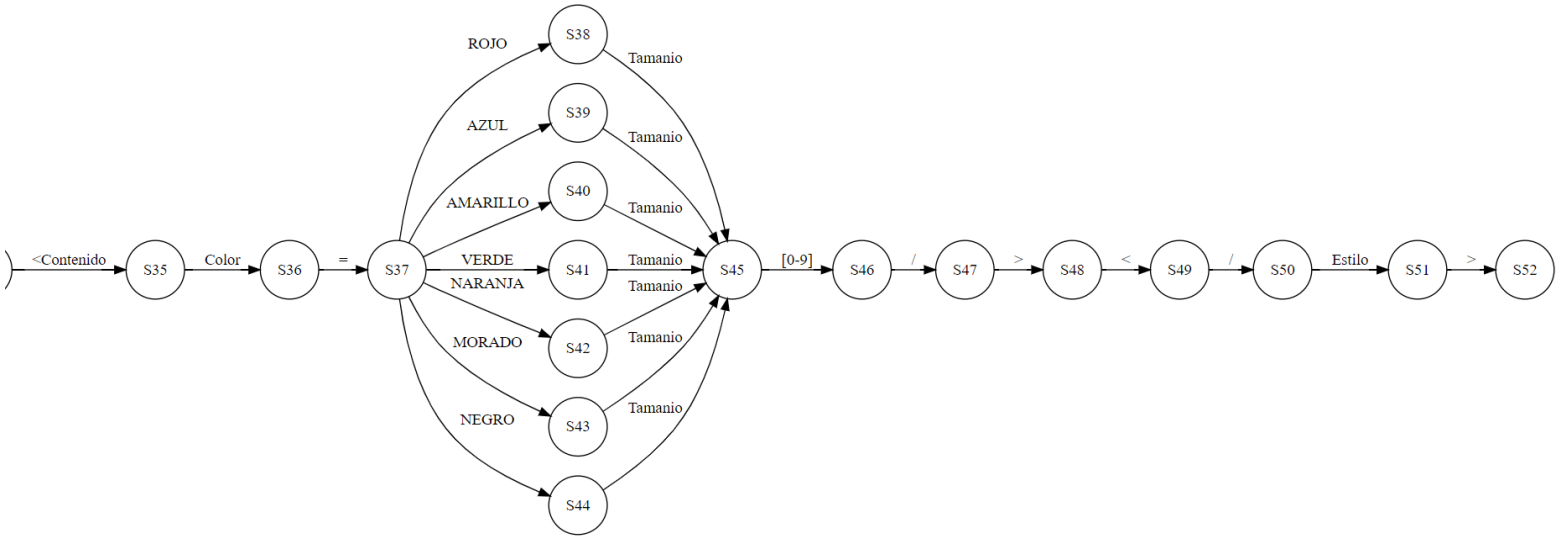
AFD de etiqueta <Función=ESCRIBIR>



AFD de etiqueta <Estilo>







Diccionario de clases:

- **GUI**
  La clase “GUI” es hereda de la clase TK de tkinter para poder construir la interfaz gráfica.

```python
class GUI(tk.Tk):
    def __init__(self):
        super().__init__()

        # Main variables
        self.title('Analizador Léxico')
        # row size
        self.rowconfigure(0,minsize=400,weight=1)
        # col size
        self.columnconfigure(1,minsize=600,weight=1)
        # Atribute with the textarea
        self.txt_area = tk.Text(self,wrap=tk.WORD)

        # This variables manage the control of the opened files
        self.file = None
        self.open_file = False

        # func to create the componentes
        self._create_components()
```

  Dicha clase cuentan con respectivos métodos para poder darle funcionalidad a los botones. En este caso los métodos que se encuentra en la imagen de abajo son específicamente para manejar el guardar, guardar como y abrir los archivos “.txt”. Además, se encuentra la funcionalidad del botón que le da toda la lógica a nuestro programa.

```python
# Func
def _open(self):
    # Opend the file window
    self.open_file = askopenfile(mode='r+')
    self.txt_area.delete(1.0,tk.END) # remove any line
    # Verified if isn't a file already opened
    if not self.open_file:
        return
    # Read the selected file
    with open(self.open_file.name, 'r+',encoding='UTF-8') as self.file:
        text = self.file.read() # read content
        self.txt_area.insert(1.0,text) # insert the text into the text area
        self.title(f'* Analizador Léxico - {self.file.name}')

def _save(self):
    # Verified if a file was already opened
    if self.open_file:
        with open(self.open_file.name,'w') as self.file:
            text = self.txt_area.get(1.0,tk.END) # read the content
            self.file.write(text) # write all the text in the textarea
            self.title(f'Analizador Léxico - {self.file.name}') # modified the title

def _save_as(self):
    # save file
    self.file = asksaveasfilename(
        defaultextension='txt',
        filetypes=[('Archivos de Texto', '*.txt'), ('Todos los archivos', '*.*')]
    )
    #if the file isn't open
    if not self.file:
        return
    with open(self.file, 'w') as file:
        text = self.txt_area.get(1.0,tk.END) # get the content
        file.write(text) # write the content in the file
        self.title(f'Analizador Léxico - {file.name}')
        self.open_file = file #Indicate that the file was already open

def _analyze(self):
    reader = LexicalAnalyzer()
    reader.compile(self.txt_area.get(1.0,tk.END))
```

Por último, se encuentran los métodos para abrir y mostrar el manual de usuario, manual técnico y la ayuda.

```python
def _manual_user(self):
    pass

def _manual_tec(self):
    pass

def _help(self):
    pass
```

- **L_Tokens**
  Esta clase hereda de la clase Enum para construir los tokens como constantes.

```python
class L_Tokens(Enum):
    # general tokens
    TK_MINOR = "<"
    TK_MAYOR = ">"
    TK_SLASH = "/"
    TK_EQUAL = "="
    TK_KEY_LEFT = "\["
    TK_KEY_RIGHT = "\]"
    # first read
    TK_E_NUMBER = "Numero"
    TK_NUMBER = "[0-9]*[.]?[0-9]+"
    # second read the operator
    TK_E_OPERATOR = "Operacion"
    TK_O_SUM = "SUMA"
    TK_O_REST = "RESTA"
    TK_O_MULT = "MULTIPLICACION"
    TK_O_DIV = "DIVISION"
    TK_O_PO = "POTENCIA"
    TK_O_SQR = "RAIZ"
    TK_O_INV = "INVERSO"
    TK_O_SEN = "SENO"
    TK_O_COS = "COSENO"
    TK_O_TAN = "TANGENTE"
    TK_O_MOD = "MOD"
    # Third read the type
    TK_TYPE = "Tipo"
    # Fourth read
    TK_E_TEXT = "Texto"
    TK_TEXT = "[A-Za-z0-9_ .,]*"
    # Fifth raed
    TK_E_FUN = "Funcion"
    TK_E_WRITE = "ESCRIBIR"
    TK_E_TITLE = "Titulo"
    TK_TITLE =  "[A-Za-z0-9_ .,]*"
    TK_E_DESCRIPTION = "Descripcion"
    TK_E_CONTENT = "Contenido"
    TK_U_TEXT = "TEXTO"
    TK_U_TYPE = "TIPO"
    # Sixth read
    TK_E_STYLE = "Estilo"
    TK_E_COLOR = "Color"
    Tk_E_SIZE = "Tamanio"
    TK_STYLE_NUMBER = "[0-9]*"
    # Colors
    TK_RED = "ROJO"
    TK_BLUE = "AZUL"
    TK_YELLOW = "AMARILLO"
    TK_GREEN = "VERDE"
    TK_ORANGE = "NARANJA"
    TK_PURPLE = "MORADO"
    TK_BLACK = "NEGRO"
```

- **LexicalAnalyzer**
  Esta clase es la que le da toda la funcionalidad y lógica a nuestro programa. Dicha clase cuenta con atributos inicializados en el constructor y diferentes métodos que se relacionan entre si para eliminar, analizar siguientes líneas, saber que etiqueta es, analizar la etiqueta número, Operador, Tipo, Función y Estilo. Donde un método compile hace que estos métodos se relacionen para luego crear un reporte o resultados.

```python
class LexicalAnalyzer:
    def __init__(self) -> None:
        self.string = "" # the character being read
        self.line = 0 # line being executed
        self.col = 0 # the column being read
        self.string_list = [] # save the values
        self.tmp_string = "" # temporary variable

        # global
        self.global_list_errors = [] # to save global errrors
        self.global_list_operations = [] # to save the operations
        self.global_list_styles = [] # save the styles
        self.dict_t_t = {} # dic to save title and text

    # Remove the character being read
    def remove(self, _string:str, _num:int):···

    # Move to the next line
    def nextLine(self):···

    # verified which label is
    def isLabel(self, _string:str, _label:str):···

    # define the analyze of <Numero>4.50</Numero>
    def Number(self, _string:str): ···

    # Operator
    def Operator(self, _string : str):···

    # The last part -> the Type
    def Type(self,_string: str):···

    # Read <Text>
    def Text(self, _string: str):···

    def Title(self, _string: str):···

    def Description(self, _string: str):···

    def Content(self, _string: str):···

    # read <Funcion=ESCRIBIR>
    def Function(self, _string: str):···

    #read <Estilo>
    def Style(self, _string: str):···

    # Method to extact the info
    def compile(self,value):···
```

- **Double_Operations**
  Al momento de leer los números de las respectivas operaciones, se crea una instancia de la clase para cada operación el cual recibe el operador y los dos números. Además, cuenta con un método el cual retorna un diccionario respectivo de la operación almacenada y el resultado.

```python
class  Double_Operations:
    def __init__(self,n1, sign , n2) -> None:

        self.n1 = float(n1)
        self.sign = sign
        self.n2 = float(n2)
        self.size = 3

    def print_operation(self):
        # print each operation
        if self.sign == "+":
            result = self.n1 + self.n2
            return {'result':str(result),'text':'Operación Suma','sign':self.sign,'n1': str(self.n1),'n2': str(self.n2)}

        elif self.sign == "-":
            result = self.n1 - self.n2
            return {'result':str(result),'text':'Operación resta','sign':self.sign,'n1': str(self.n1),'n2': str(self.n2)}

        elif self.sign == "*":
            result = self.n1 * self.n2
            return {'result':str(result),'text':'Operación multiplicacion','sign':self.sign,'n1': str(self.n1),'n2': str(self.n2)}

        elif self.sign == "/":
            try:
                result = self.n1 / self.n2
                return {'result':str(result),'text':'Operación división','sign':self.sign,'n1': str(self.n1),'n2': str(self.n2)}
            except ZeroDivisionError :
                return {'result':'No se puede efectuar la división','text':'Operación multiplicacion','sign':self.sign,'n1': str(self.n1),'n2': str(self.n2)}

        elif self.sign == "**":
            result = self.n2 ** self.n1
            return {'result':str(result),'text':'Operación potencia','sign':self.sign,'n1': str(self.n2),'n2': str(self.n1)}

        elif self.sign == "mod":
            result = self.n1 % self.n2
            return {'result':str(result),'text':'Operación mod','sign':'%','n1': str(self.n1),'n2': str(self.n2)}

        elif self.sign == 'sqr':
            result = (self.n2)**(1/self.n1)
            return {'result':str(result), 'text': 'Operación raiz', 'sign': self.sign, 'n1': str(self.n1),'n2': str(self.n2)}
```

- **Single_Operations**
  Al igual que la primera clase, tiene la misma funcionalidad, con la peculiaridad que solo recibe un operador y un número, donde su método retorna un diccionario con sus resultado y datos utilizados.

```python
class  Double_Operations:
    def __init__(self,n1, sign , n2) -> None:

        self.n1 = float(n1)
        self.sign = sign
        self.n2 = float(n2)
        self.size = 3

    def print_operation(self):
        # print each operation
        if self.sign == "+":
            result = self.n1 + self.n2
            return {'result':str(result),'text':'Operación Suma','sign':self.sign,'n1': str(self.n1),'n2': str(self.n2)}

        elif self.sign == "-":
            result = self.n1 - self.n2
            return {'result':str(result),'text':'Operación resta','sign':self.sign,'n1': str(self.n1),'n2': str(self.n2)}

        elif self.sign == "*":
            result = self.n1 * self.n2
            return {'result':str(result),'text':'Operación multiplicacion','sign':self.sign,'n1': str(self.n1),'n2': str(self.n2)}

        elif self.sign == "/":
            try:
                result = self.n1 / self.n2
                return {'result':str(result),'text':'Operación división','sign':self.sign,'n1': str(self.n1),'n2': str(self.n2)}
            except ZeroDivisionError :
                return {'result':'No se puede efectuar la división','text':'Operación multiplicacion','sign':self.sign,'n1': str(self.n1),'n2': str(self.n2)}

        elif self.sign == "**":
            result = self.n2 ** self.n1
            return {'result':str(result),'text':'Operación potencia','sign':self.sign,'n1': str(self.n2),'n2': str(self.n1)}

        elif self.sign == "mod":
            result = self.n1 % self.n2
            return {'result':str(result),'text':'Operación mod','sign':'%','n1': str(self.n1),'n2': str(self.n2)}

        elif self.sign == 'sqr':
            result = (self.n2)**(1/self.n1)
            return {'result':str(result), 'text': 'Operación raiz', 'sign': self.sign, 'n1': str(self.n1),'n2': str(self.n2)}
```

- **Lexical_Errors**
  Esta clase crea objetos de los respectivos errores léxicos encontrados en la lectura del archivo. Almacena el valor donde ocurrió el error, columna, fila y el tipo de error el cual retorna un diccionario con su método respectivo.

```python
class Lexical_Errors:

    #Constructor
    def __init__(self,_token, _row, _col, _descrip = "") -> None:
        self._token = _token
        self._row = _row
        self._col = _col
        self._descrip = _descrip

    def getError(self):
        return {'token':self._token, 'row': self._row, 'col': self._col , 'description': self._descrip}
```

- **Styles**
  Esta clase crea objetos de tipo estilo donde se almacena su color y tamaño que. Donde su método retorna un diccionario con sus respectivos valores.

```python
class Styles:
    def __init__(self, label, color, size ) -> None:
        self.label = label
        self.color = color
        self.size = size

    def getStyles(self):
        color = None
        if self.color == 'ROJO':
            color = 'red'
        elif self.color == 'AZUL':
            color = 'blue'
        elif self.color == 'AMARILLO':
            color = 'yellow'
        elif self.color == 'VERDE':
            color = 'green'
        elif self.color == 'NARANJA':
            color = 'orange'
        elif self.color == 'MORADO':
            color = 'purple'
        elif self.color == 'NEGRO':
            color = 'black'
        else:
            color = 'black'
        return {'label':self.label, 'color': color, 'size':self.size}
```

Diccionario de métodos y funciones:

- **generateHTMLResult**
  Este método recibe la lista de operaciones, lista de estilos, titulo y texto los cuales crean y le dan estilos a un HTML el cual contiene el resultado de todas las operaciones efectuadas para generar un archivo HTML.

```python
def generateHTMLResult(list_r, list_s, title="Titulo", text="Texto"):
    html = f'''
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>PROYECTO_1_LFP</title>
</head>
<style>
```

```
    # styles
    for st in list_s:
        dict = st.getStyles()
        if dict['label'] == "Titulo":
            str = '''
            h1 {

            '''
            style = f'''
            color:{dict["color"]};
            font-size:{dict["size"]}px;
            '''
            str += f'''
            {style}
            '''
            str += '''
            }
            '''
            html += str
        elif dict['label'] == "Descripcion":
            str = '''
            h3 {

            '''
            style = f'''
            color:{dict["color"]};
            font-size:{dict["size"]}px;
            '''
            str += f'''
            {style}
            '''
            str += '''
            }
            '''
            html += str

        elif dict['label'] == "Contenido":
            str = '''
            p {

            '''
            style = f'''
            color:{dict["color"]};
            font-size:{dict["size"]}px;
            '''
            str += f'''
            {style}
            '''
            str += '''
            }
            '''
            html += str

    html += '''
</style>
    '''
```

```
<body>
    <h1>{title}</h1>
    <h3>{new_text6}</h3>
    '''
    # Made the operations
    html_operations = """
    <div>
    """
    for op in list_r:
        if op.size == 3:
            dict = op.print_operation()
            if dict['sign'] == "sqr":
                operation = f'''
                <p>
                    {dict["text"]}: <br>
                    <br>
                    ({dict["n2"]})^(1/{dict["n1"]}) = {dict["result"]}
                </p>
                '''
            else:
                operation = f'''
                <p>
                    {dict["text"]}: <br>
                    <br>
                    {dict["n1"]} {dict["sign"]} {dict["n2"]} = {dict["result"]}
                </p>
                '''
            html_operations += operation

        elif op.size == 2:
            dict = op.print_operation()
            if dict['sign'] == "inv":
                operation = f'''
                <p>
                    {dict["text"]}: <br>
                    <br>
                    1/{dict["n"]} = {dict["result"]}
                </p>
                '''
            else:
                operation = f'''
                <p>
                    {dict["text"]}: <br>
                    <br>
                    {dict["sign"]}({dict["n"]})  = {dict["result"]}
                </p>
                '''
            html_operations += operation

        html_operations += """
    </div>
    """
    html += html_operations
    html += '''
</body>
    '''
    document = open('response/RESULTADOS_202109567.html','w')
    document.write(html)
    document.close()
```

- **errorsHTML**
  Esta función recibe una lista de errores, donde crea una tabla con la herramienta graphviz, donde posteriormente se convierte en una imagen png para ser utilizada en un HTML que contiene como título errores y la imagen de la tabla hecha con Graphviz.

```
def errorsHTML(list_e):
    graph = '''
digraph  ERRORES {
    node[shape=plaintext];

    tabla[label =<
    <TABLE>
        <TR>
            <td>No.</td>
            <td>Lexema</td>
            <td>Tipo</td>
            <td>Columna</td>
            <td>Fila</td>
        </TR>
    '''
    for i in range(len(list_e)):
        object = list_e[i]
        dict = object.getError()
        sign = ""
        if dict['token'] == ">":
            sign = "&gt;"
        elif dict['token'] == "<":
            sign = "&lt;"
        else:
            sign = dict['token']

        err = f'''
        <TR>
            <TD>{i+1}</TD>
            <TD>{sign}</TD>
            <TD>{dict['description']}</TD>
            <TD>{dict['col']}</TD>
            <TD>{dict['row']}</TD>
        </TR>
        '''
        graph += err

    graph += '''
    </TABLE>
    >];
}
    '''
    # write the dot
    with open('helpers/table.txt','w') as file:
        file.write(graph)

    os.system('dot.exe -Tpng helpers/table.txt -o response/img/tableE.png')
```

```
    # Generate the HTML
    html = f'''
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>PROYECTO_1_LFP</title>
</head>
<body>
    <h1>Errores</h1>
    <img src="img/tableE.png">
</body>
    '''

    document = open('response/ERRORES_202109567.html','w')
    document.write(html)
    document.close()
```