MANUAL TÉCNICO

[PROYECTO 1]

DAVED ABSHALON EJCALON CHONAY - 202105668

INGENIERÍA EN CIENCIAS Y SISTEMAS

LENGUAJES FORMALES Y DE POGRAMACIÓN, B+

REQUISITOS

- Sistema operativo
  Windows 10

- Procesador
  Intel Core 2 Duo E8200, AMD Phenom II X6 1075 a 3,0 GHz

- Memoria
  1 GB de RAM

- Almacenamiento
  1 GB de espacio disponible

- Tarjeta gráfica
  NVIDIA GT 630 / AMD HD 7750

- Internet
  Acceso a internet, con cualquier explorador web

- Visual Studio
  Con python instalado

DESARROLLO DEL PROGRAMA

- Carpeta principal

Archivos de código fuente: Aquí se encuentran todos los archivos Python que componen el programa.

Documentación: Carpeta opcional que contiene documentos como este manual técnico.

Paradigma de programación utilizado

Principalmente programación orientada a objetos (POO) donde los conceptos claves del programa son las clases y objetos.

Clases y objetos: Este programa está organizado por clases y objetos en los cuales están encapsulados el conjunto de datos y funciones relacionadas que representan el código unificado del programa. (Carga, Gestion, Filtrado, Grafica y la clase Main).

Encapsulación: Cada clase tiene su propia funcionalidad en la cual todas funcionan en conjunto y cada detalle interno se necesita uno del otro.

Abstracción: Todas las clases proporcionan una abstracción al mundo real como lo son las películas y actores, esto permite una mayor compresión por parte del programador.

Este manual técnico proporciona una descripción detallada del código Python que implementa un analizador léxico. El analizador léxico es capaz de reconocer tokens válidos y erróneos en un archivo de entrada y generar archivos HTML que muestran estos tokens.

CLASES

- Lexico.py

El código Python proporcionado consta de una clase Lexer que implementa el analizador léxico. Aquí hay una breve descripción de los componentes clave del código:

- Lexer

La clase principal que contiene los métodos para analizar el archivo de entrada, identificar tokens válidos y erróneos, y generar archivos HTML con los resultados.

```
class Lexer:
    def __init__(self):
        self.tabla_tokens_validos = []
        self.tabla_tokens_erroneos = []
        self.cadena = ''
        self.fila = 0
        self.columna = 0
```

- Metodo analizar

Este método analiza la cadena de entrada y utiliza un autómata finito determinista (DFA) para reconocer tokens válidos y erróneos.

Este código es un analizador léxico que escanea una cadena de entrada y produce tokens basados en ciertas reglas definidas. Aquí hay un resumen técnico:

- El método analizar toma una cadena como entrada y realiza un análisis léxico sobre ella.
- Inicializa algunas variables de estado, como fila, columna, estado_actual y lexema.
- Itera sobre cada caracter en la cadena.
- Implementa una máquina de estados finitos para reconocer diferentes tipos de tokens.

- Los estados se definen como:

1. Estado 0: Estado inicial. Evalúa el tipo de caracter y cambia de estado según las reglas.
2. Estado 1: Reconoce identificadores.
3. Estado 2: Reconoce números.
4. Estado 3: Reconoce cadenas entre comillas dobles.
5. Estado 4: Reconoce identificadores entre paréntesis.

```
def analizar(self, cadena):
    self.cadena = cadena
    self.fila = 1
    self.columna = 0
    estado_actual = 0
    lexema = ''

    for caracter in cadena:
        if caracter == '\n':
            self.fila += 1
            self.columna = 0
            continue
        self.columna += 1
```

- Se utilizan diferentes acciones basadas en el estado actual y el tipo de caracter.
- Los tokens válidos se guardan usando el método guardar_token_valido.
- Los errores se manejan utilizando el método recuperar_error.
- El análisis se completa cuando se alcanza el final de la cadena.
- Se proporciona un manejo especial para cadenas vacías, cadenas numéricas y cierre de paréntesis sin identificador.

```
if estado_actual == 0:
    if caracter.isspace():
        continue
    elif caracter.isalpha() or caracter == '_':
        estado_actual = 1
        lexema += caracter
    elif caracter == '{':
        self.guardar_token_valido('LLAVE_IZQ', caracter)
    elif caracter == '}':
        self.guardar_token_valido('LLAVE_DER', caracter)
    elif caracter == ':':
        self.guardar_token_valido('DOS_PUNTOS', caracter)
    elif caracter == ';':
        self.guardar_token_valido('PUNTO_COMA', caracter)
    elif caracter == ',':
        self.guardar_token_valido('COMA', caracter)
    elif caracter == '[':
        self.guardar_token_valido('CORCHETE_IZQ', caracter)
    elif caracter == ']':
        self.guardar_token_valido('CORCHETE_DER', caracter)
    elif caracter == '=':
        self.guardar_token_valido('IGUAL', caracter)
    elif caracter == '"':
        estado_actual = 3
    elif caracter.isdigit():
        estado_actual = 2
        lexema += caracter
    elif caracter == '(':
        estado_actual = 4
    else:
        self.recuperar_error(caracter)
```

- Metodos de cada tokens

Estas son las funciones adicionales que manejan la gestión de tokens válidos y tokens erróneos en el código:

1. guardar_token_valido: Esta función recibe un tipo de token y su valor como entrada, y luego agrega un diccionario a la lista tabla_tokens_validos, que contiene información sobre el tipo de token, su valor, así como la fila y la columna donde se encontró en la cadena de entrada.
2. guardar_token_erroneo: Similar a la función anterior, esta función guarda información sobre caracteres erróneos encontrados en la cadena de entrada. Crea un diccionario con el caracter erróneo, junto con la fila y columna donde se encontró, y lo agrega a la lista tabla_tokens_erroneos.
3. recuperar_error: Esta función se invoca cuando se encuentra un carácter no válido en la cadena de entrada. Guarda el carácter erróneo utilizando la función guardar_token_erroneo, lo que permite un seguimiento de los errores. También podrías descomentar la línea que imprime un mensaje de error para obtener información sobre el carácter no válido.

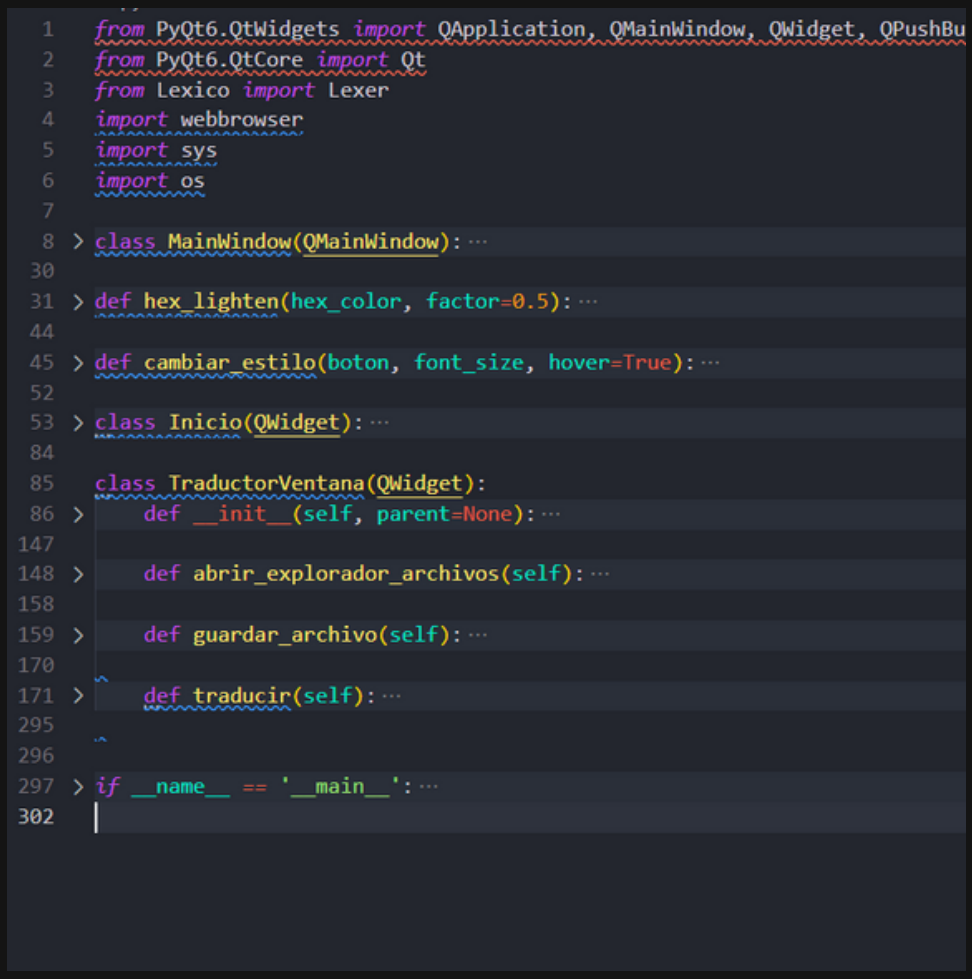
```
def guardar_token_valido(self, tipo, valor):
    self.tabla_tokens_validos.append({'tipo': tipo, 'valor': valor, 'fila': self.fila, 'columna': self.columna})

def guardar_token_erroneo(self, caracter):
    self.tabla_tokens_erroneos.append({'caracter': caracter, 'fila': self.fila, 'columna': self.columna})

def recuperar_error(self, caracter):

    self.guardar_token_erroneo(caracter)
```

- Interfaz.py

Este código es una aplicación de escritorio desarrollada con PyQt6 que actúa como un traductor HTML. Aquí tienes un resumen técnico:

1. **Clases Principales:**
   - MainWindow: Es la ventana principal de la aplicación. Contiene un widget apilado (QStackedWidget) que permite cambiar entre la pantalla de inicio y la pantalla del traductor.
   - Inicio: Representa la pantalla de inicio de la aplicación. Contiene un botón para continuar al traductor y otro para salir de la aplicación.
   - TraductorVentana: Representa la pantalla del traductor HTML. Permite al usuario ingresar texto, traducirlo y guardar el resultado en un archivo HTML. También muestra los tokens analizados y los tokens erróneos en formato HTML.
2. **Funciones Útiles:**
   - hex_lighten: Toma un color en formato hexadecimal y devuelve un color más claro.
   - cambiar_estilo: Cambia el estilo de un botón, oscureciéndolo o aclarándolo cuando el mouse entra o sale de él.
3. **Funciones de Interfaz de Usuario:**
   - abrir_explorador_archivos: Abre un explorador de archivos para que el usuario seleccione un archivo para traducir.
   - guardar_archivo: Guarda el contenido traducido en un archivo HTML.
   - traducir: Realiza la traducción del texto ingresado utilizando un analizador léxico y genera archivos HTML con los tokens analizados y erróneos.
4. **Módulos Importados:**
   - QApplication, QMainWindow, QWidget, QPushButton, QLabel, QHBoxLayout, QVBoxLayout, QTextEdit, QFileDialog, QMessageBox: Son clases de PyQt6 para crear la interfaz de usuario y manejar eventos.
   - Qt: Proporciona constantes y enums utilizados por Qt.
   - webbrowser: Permite abrir archivos HTML en el navegador predeterminado.
   - sys, os: Proporcionan funcionalidades del sistema operativo, como cerrar la aplicación y manipular archivos y rutas.
5. **Ejecución del Programa:**
   - Se crea una instancia de QApplication y se inicia el bucle de eventos de la aplicación.
   - Se crea una instancia de MainWindow y se muestra.
   - La aplicación permanece en ejecución hasta que se cierra la ventana principal.

En resumen, este código implementa una aplicación de traducción HTML con una interfaz gráfica de usuario utilizando PyQt6, que permite a los usuarios ingresar texto, traducirlo y guardar el resultado en archivos HTML.



```
from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget, QPushBu
from PyQt6.QtCore import Qt
from Lexico import Lexer
import webbrowser
import sys
import os

class MainWindow(QMainWindow):···

def hex_lighten(hex_color, factor=0.5):···

def cambiar_estilo(boton, font_size, hover=True):···

class Inicio(QWidget):···

class TraductorVentana(QWidget):
    def __init__(self, parent=None):···

    def abrir_explorador_archivos(self):···

    def guardar_archivo(self):···

    def traducir(self):···

if __name__ == '__main__':···
```

```
class Inicio(QWidget):
    def __init__(self, parent=None):
        super(Inicio, self).__init__(parent)
        self.setWindowTitle("Editor HTML")

        self.etiqueta_curso_seccion = QLabel(
            "Daved Abshalon Ejcalon Chonay\nCurso: Lenguajes Formales y de Programación\nSección: B+"
        )
        self.etiqueta_curso_seccion.setStyleSheet("color: #FFFFFF; font-size: 15px; font-weight: bold;")
        self.etiqueta_curso_seccion.setAlignment(Qt.AlignmentFlag.AlignCenter)

        self.boton_salir = QPushButton("Salir")
        self.boton_salir.clicked.connect(self.salir)

        self.boton_continuar = QPushButton("Continuar")

        for btn in [self.boton_salir, self.boton_continuar]:
            btn.setStyleSheet("background-color: #616161; color: #FFFFFF; font-size: 12px")
            btn.setCursor(Qt.CursorShape.PointingHandCursor)
            btn.enterEvent = lambda event, boton=btn: cambiar_estilo(boton, 12, True)
            btn.leaveEvent = lambda event, boton=btn: cambiar_estilo(boton, 12, False)

        layout = QVBoxLayout()
        layout.addWidget(self.etiqueta_curso_seccion)
        layout.addWidget(self.boton_continuar)
        layout.addWidget(self.boton_salir)

        self.setLayout(layout)

    def salir(self):
        sys.exit()
```