Universidad de Guadalajara

Centro Universitario

De

Ciencias

ExactasE

Ingenierías

(CUCEI)

Materia:

SEMINARIO DE SOLUCION DE PROBLEMAS

DETRADUCTORES DE LENGUAJES II

Nombre: Aceves López

JonathanCódigo:

217481363

Main:

```python
from PySide6.QtWidgets import QApplication
from mainwindow import MainWindow
import sys

#Aplicacion de Qt
app = QApplication()
#Invocacion de la ventana principal
window = MainWindow()
#Se hace visible la ventana
window.show()
# Qt Loop
sys.exit(app.exec())
```

Esta parte del Código utiliza “PySide6” para crear una aplicación de escritorio con interfaz gráfica

1.- from PySide6.QtWidgets import QApplication: Importa la clase QApplication del módulo QtWidgets de PySide6. QApplication es una clase fundamental en las aplicaciones Qt, ya que se encarga de manejar el flujo de ejecución de la aplicación y administrar los recursos.

2.- from mainwindow import MainWindow: Importa la clase MainWindow desde un módulo llamado mainwindow. Presumiblemente, MainWindow es la clase que define la ventana principal de la aplicación.

3.- import sys: Importa el módulo sys, que proporciona acceso a algunas variables y funciones utilizadas o mantenidas por el intérprete de Python.

4.- app = QApplication(): Crea una instancia de la clase QApplication, que representa la aplicación Qt. Es importante tener exactamente una instancia de QApplication en cada aplicación Qt.

5.- window = MainWindow(): Crea una instancia de la clase MainWindow, representa la ventana principal de la aplicación. Esta clase probablemente define la interfaz de usuario y las interacciones asociadas con la ventana principal de la aplicación.

6.- window.show(): Muestra la ventana principal de la aplicación en pantalla.

7.- sys.exit(app.exec()): Este es el bucle principal de eventos de Qt. app.exec() inicia el bucle de eventos de la aplicación Qt y devuelve el estado de salida al finalizar. sys.exit() asegura que cuando la aplicación finalice, la ejecución del script también finalice correctamente.

Mainwindow:

Realizar análisis léxico y sintáctico de un lenguaje de programación. La aplicación está desarrollada en Python utilizando la biblioteca PySide6 y la interfaz gráfica Qt.

Funcionalidades:

Análisis léxico: Obtiene los tokens del código fuente y los muestra en una tabla con el nombre del token, el lexema y la posición.
Análisis sintáctico: Verifica la corrección sintáctica del código fuente y muestra los errores encontrados en un cuadro de texto.
Abrir archivo: Permite abrir un archivo de texto con código fuente para analizar.
Guardar archivo: Permite guardar el código fuente en un archivo de texto.
Análisis del código:

Estructura: El código está organizado en una clase principal MainWindow que hereda de QMainWindow. La clase contiene los métodos para la gestión de la interfaz gráfica y la funcionalidad del análisis léxico y sintáctico.
Modularidad: El código está dividido en módulos independientes para el análisis léxico (lexico.py) y sintáctico (sintactico.py).
Manejo de errores: Se capturan las excepciones y se muestran mensajes de error al usuario.

Léxico:

El analizador léxico identifica y clasifica los tokens (elementos léxicos básicos) en el código fuente.

Componentes:

1.- Enum TokenType: Define un enumerador TokenType para representar los diferentes tipos de tokens posibles en el lenguaje.
2.- Clase Token: Define la clase Token para almacenar la información de un token, incluyendo su tipo, lexema (cadena de texto del token) y la línea del código fuente en la que se encuentra.
3- Diccionario token_types: Asocia el valor textual de cada token con una tupla que contiene su nombre descriptivo y una descripción breve.
4.- Funciones:

- is_operator: Determina si un caracter es un operador aritmético o relacional.
- is_separator: Determina si un caracter es un separador (corchetes, llaves, paréntesis, coma, punto y coma).
- valid_number: Valida si una cadena representa un número entero o un número flotante.
- valid_id: Valida si una cadena es un identificador válido (inicia con letra o guión bajo y seguido de letras, números o guiones bajos).
- split_code: Divide el código fuente en lexemas individuales considerando las cadenas literales y separadores.
- get_tokens: Obtiene los tokens del código fuente. Divide el código en lexemas usando split_code y luego clasifica cada lexema en su tipo de token correspondiente.

Funcionamiento:

1.- La función get_tokens recibe el código fuente como entrada.
2.- La función split_code primero divide el código fuente en lexemas individuales, considerando cadenas literales y separadores.
3.- La función get_tokens itera sobre cada lexema:

- Busca el lexema en el diccionario token_types. Si se encuentra, se crea un token del tipo correspondiente.
- Si no se encuentra en el diccionario y es una cadena literal, se clasifica como token de tipo cadena.
- Si no se encuentra y es un identificador válido, se clasifica como token de tipo identificador.
- Si no se encuentra en ninguna categoría anterior, se valida si es un número entero o flotante. Si lo es, se crea un token del tipo correspondiente.
- Si no se puede clasificar en ninguna de las categorías anteriores, se crea un token de tipo desconocido.

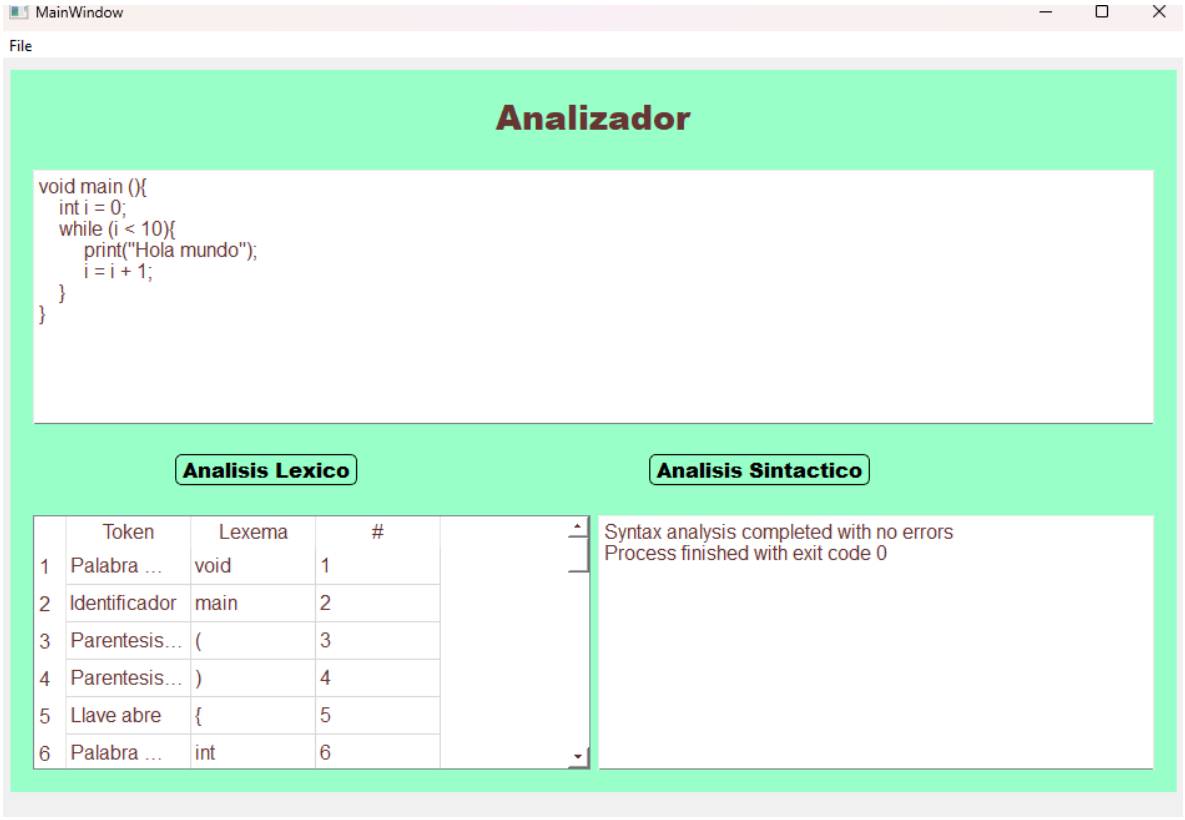
4.- La función devuelve una lista de tokens.

Sintáctico:
El analizador semántico verifica la validez semántica del código fuente, como la tipificación de variables, la estructura de las instrucciones y la compatibilidad de tipos en operaciones.

Componentes:

- Módulo semantico: Contiene las funciones para el análisis semántico.
- Diccionario token_types: Asocia el valor textual de cada token con su nombre descriptivo y una descripción breve.
- Variable: Clase para almacenar información de una variable, como su nombre, tipo y valor.
- Funciones:

programa: Función principal que inicia el análisis del código fuente.
declaraciones: Analiza las declaraciones de variables.
instrucciones: Analiza las instrucciones del programa.
asignacion: Analiza la asignación de un valor a una variable.
exprAritmetica: Analiza una expresión aritmética.
comparacion: Analiza una comparación entre dos valores.
opComparacion: Valida el operador de comparación utilizado.
operador: Valida un operador aritmético o de comparación.
buscarVar: Busca una variable en la tabla de símbolos.
buscarValor: Obtiene el valor de una variable.


MainWindow
File
Analizador
void main (){
int i = 0;
while (i < 10){
print("Hola mundo");
i = i + 1;
}
}
Analisis Lexico
Analisis Sintactico
Token
Lexema
#
1 Palabra ... void 1
2 Identificador main 2
3 Parentesis... (3
4 Parentesis...) 4
5 Llave abre { 5
6 Palabra ... int 6
Syntax analysis completed with no errors
Process finished with exit code 0