Manual De Técnico

02/05/2024

Samuel Nehemias Coyoy Perez

202200198

Contenido

[Objetivos 3](#_Toc165509704)

[Específicos 3](#_Toc165509705)

[Generales 3](#_Toc165509706)

[Especificación Técnica 3](#_Toc165509707)

[Requisitos Hardware 3](#_Toc165509708)

[Requisitos Software 3](#_Toc165509709)

[Manual Técnico: Editor de Texto con Funcionalidades Especiales 4](#_Toc165509710)

[Analizador.py 4](#_Toc165509711)

[Clase Errores.py 7](#_Toc165509712)

[Funcion 7](#_Toc165509713)

[Editor de texto 8](#_Toc165509714)

[Método del árbol 10](#_Toc165509715)

[Automata 11](#_Toc165509716)

[Tokens 11](#_Toc165509717)

[Gramática libre de contexto 12](#_Toc165509718)

# Objetivos

## Específicos

Visualizacion de tokens

## Generales

Creación de mongoDB

Verificación de errores

# Especificación Técnica

## Requisitos Hardware

Mouse

Teclado

Monitor

## Requisitos Software

Sistema Operativo: Windows 10

Herramientas: Visual Studio Code

Lenguaje de Programacion: Python

# Manual Técnico: Editor de Texto con Funcionalidades Especiales

# Analizador.py

Todo inicia desde acá, teniendo los lexemas y los errores auxiliando a que todo salga bien, el inicio de este proceso iniciaría al mandar datos a la función de instrucciones desde el editor de texto.

1. **Importaciones**:
   * El código comienza importando algunas funciones o clases desde módulos específicos. Por ejemplo, **Instrucciones.Funcion**, **Error.Errores**, y **Abstract.Lexema**. Esto significa que el código utiliza funcionalidades definidas en estos módulos.
2. **Variables globales**:
   * Se definen varias variables globales como **n\_linea**, **n\_columna**, **instrucciones**, **lista\_lexemas**, y **lista\_errores**. Estas se utilizan para almacenar información relevante durante el análisis de las instrucciones.
3. **Función instruccion**:
   * Esta función recibe una cadena de texto (**cadena**) que representa una instrucción y la procesa.
   * Itera sobre cada carácter de la cadena, realizando diferentes acciones dependiendo del tipo de carácter.
   * Por ejemplo, si el carácter es una letra, se arma un lexema y se agrega a **lista\_lexemas**.
   * También maneja caracteres como '=' para formar lexemas y otros como '\t', '\n', ' ', '\r', '-', '/\* '.
   * Si encuentra un carácter que no puede manejar, lo considera un error léxico y lo agrega a **lista\_errores**.
4. **Función armar\_lexema**:
   * Esta función ayuda en la creación de lexemas. Toma una cadena de texto y devuelve el lexema creado y la cadena restante después de procesarlo.
5. **Función operar**:
   * Esta función parece ser la encargada de realizar operaciones específicas basadas en los lexemas previamente analizados.
   * Itera sobre los lexemas en **lista\_lexemas** y realiza operaciones dependiendo de su tipo.
   * Cada tipo de operación parece corresponder a un tipo de instrucción específica, como crear o eliminar una base de datos, una colección, insertar datos, actualizar datos, eliminar datos, buscar datos, etc.
   * Cada vez que encuentra un lexema relevante para una operación, lo saca de **lista\_lexemas** y crea una instancia de la clase **Funcion** con los argumentos necesarios.
   * Esta función parece un poco repetitiva y podría refactorizarse para eliminar la duplicación de código.
6. **Función operar\_**:
   * Esta función parece ser una versión modificada de **operar**. Itera llamando a **operar** y almacenando los resultados en **instrucciones**.
7. **Función getErrores**:
   * Simplemente devuelve la lista de errores encontrados durante el análisis

global n\_linea

global n\_columna

global instrucciones

global lista\_lexemas

global lista\_errores

n\_linea = 1

n\_columna = 1

lista\_lexemas = []

instrucciones = []

lista\_errores = []

def instruccion(cadena):

    return lista\_lexemas

def armar\_lexema(cadena):

    global n\_linea

    global n\_columna

    global lista\_lexemas

    lexema = ''

    puntero = ''

    for char in cadena:

        puntero += char

        if char == ' ' or char == ';':

            return lexema, cadena[len(puntero):]    #! si encuentra una  " termino de leer el token

        else:

            lexema += char   #! creamos nuestros Token

    return None, None

def operar():

    global lista\_lexemas

    global instrucciones

    while lista\_lexemas:

        lexema = lista\_lexemas.pop(0)

        if lexema.operar(None) == 'CrearBD':

            nombredb = lista\_lexemas.pop(0)

            igualdb = lista\_lexemas.pop(0)

            nuevadb = lista\_lexemas.pop(0)

            creardb2 = lista\_lexemas.pop(0)

            func =  Funcion(lexema.lexema, nombredb.lexema, igualdb.lexema, nuevadb.lexema, creardb2.lexema, lexema.getFila(), lexema.getColumna())

    return None

def operar\_():

    global instrucciones

    temp\_instrucciones = []

    while True:

        operacion = operar()

        if operacion:

            temp\_instrucciones.append(operacion)

        else:

            break

    instrucciones = temp\_instrucciones

    return instrucciones

def getErrores():

    global lista\_errores

    return lista\_errores

# Clase Errores.py

La clase Errores representa un error en el código fuente. Al crear una instancia de esta clase, se proporciona el lexema problemático, el tipo de error y su ubicación en el código fuente. La clase tiene un método para obtener una representación del error y métodos para obtener las coordenadas de este.

from Abstract.Abstract import Expression

class Errores(Expression):

    def \_\_init\_\_(self, lexema, tipo, fila, columna):

        self.lexema = lexema

        self.tipo = tipo

        super().\_\_init\_\_(fila, columna)

    def operar(self, no):

        lex = "Error: " + self.lexema

        return lex

    def getColumna(self):

        return super().getColumna()

    def getFila(self):

        return super().getFila()

# Funcion

1. **Inicialización**:
   * El constructor **\_\_init\_\_** inicializa los atributos de la instancia con los valores proporcionados.
2. **Método ejecutarT**:
   * Este método evalúa la operación representada por la instancia y devuelve una cadena de texto que corresponde a la operación ejecutada en el contexto de una base de datos MongoDB.
3. **Método listaTokens**:
   * Este método genera una lista de tokens encontrados en la operación representada por la instancia.
4. **Método listaErrores**:
   * Este método genera una lista de errores encontrados en la operación representada por la instancia.
5. **Métodos getFila y getColumna**:
   * Estos métodos devuelven las coordenadas de fila y columna de la instancia.

class Funcion(Expression):

    def \_\_init\_\_(self, crear, nombre, igual, nueva, crear2, fila, columna):

        # Inicializa los atributos de la instancia

    def ejecutarT(self):

        # Ejecuta la operación representada por la instancia y devuelve el resultado como cadena de texto

    def listaTokens(self):

        # Genera una lista de tokens encontrados en la operación representada por la instancia

    def listaErrores(self):

        # Genera una lista de errores encontrados en la operación representada por la instancia

    def getFila(self):

        # Devuelve la coordenada de fila de la instancia

    def getColumna(self):

        # Devuelve la coordenada de columna de la instancia

# Editor de texto

1. **Interfaz de usuario de editor de texto**: La aplicación muestra una ventana con un área de edición de texto donde los usuarios pueden escribir y editar texto.
2. **Funciones de archivo**: Permite a los usuarios abrir, guardar, guardar como y crear nuevos archivos de texto.
3. **Función de análisis**: Hay una función para analizar el contenido del editor de texto. Esta función analiza el texto utilizando un analizador léxico y sintáctico, luego muestra los resultados del análisis en una nueva ventana.
4. **Funciones de tokens:** En base a los datos que se tienen del análisis se debe de generar un texto de los tokens.
5. **Funciones de errores**: Los errores se mostrarán en una ventana diferente y se dirá cuál es el error y que esperaba el programa.
6. **Números de línea**: El editor de texto incluye una barra de números de línea que muestra el número de línea correspondiente a cada línea en el área de edición de texto.

# Función para ejecutar la aplicación

def ejecutar():

    # Configurar y ejecutar la aplicación de edición de texto

    root = tk.Tk()

    app = TextEditorApp(root)

    root.mainloop()

# Clase para la aplicación de edición de texto

class TextEditorApp:

    def \_\_init\_\_(self, root):

        # Inicializar la interfaz de usuario

        self.configurar\_interfaz()

    # Funciones para acciones de archivo: nuevo, abrir, guardar, guardar como

    def nuevo\_archivo(self):

        pass

    def abrir\_archivo(self):

        pass

    def guardar\_archivo(self):

        pass

    def guardar\_archivo\_como(self):

        pass

    # Función para analizar el texto y mostrar los resultados

    def analizar\_texto(self):

        pass

    # Función para mostrar los tokens encontrados durante el análisis

    def mostrar\_tokens(self):

        pass

    # Función para mostrar los errores encontrados durante el análisis

    def mostrar\_errores(self):

        pass

    # Función para actualizar los números de línea en respuesta a eventos

    def actualizar\_numeros\_linea(self, event=None):

        pass

    # Función para configurar la interfaz de usuario

    def configurar\_interfaz(self):

        pass

# Función principal para ejecutar la aplicación

ejecutar()

# Método del árbol

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

# Automata

Diagrama

Descripción generada automáticamente

# Tokens

|  |  |  |
| --- | --- | --- |
| Token | Descripcion | Patron |
| PalabraReservada | Palabra utilizada para generar CrearBD | CrearBD |
| PalabraReservada | Palabra utilizada para generar EliminarBD | EliminarBD |
| PalabraReservada | Palabra utilizada para generar CrearColeccion | CrearColeccion |
| PalabraReservada | Palabra utilizada para generar EliminarColeccion | EliminarColeccion |
| PalabraReservada | Palabra utilizada para generar InsertarUnico | InsertarUnico |
| PalabraReservada | Palabra utilizada para generar ActualizarUnico | ActualizarUnico |
| PalabraReservada | Palabra utilizada para generar EliminarUnico | EliminarUnico |
| PalabraReservada | Palabra utilizada para generar BuscarUnico | BuscarUnico |
| PalabraReservada | Palabra utilizada para generarBuscarTodo | BuscarTodo |
| PalabraNueva | Para decir que queremos crear una nueva funcion | nueva |
| Identificador | Nombre de nuestras funciones | [a-zA-Z]][a-zA-Z0-9] |
| LlavApe | Llave apertura | { |
| Llavcerr | Llave cerrar | } |
| Pareapertu | Parentesis apertura | ( |
| Parecerr | Parentesis cerrar | ) |
| Coma | Coma | , |
| IguaI | Igual | = |
| puntco | punto y coma | ; |
| Dos punt | Dos puntos | : |
| Comidobl | Comillas dobles | " |
| ComillaSimpl | Comilla simple | ‘ |

# Gramática libre de contexto

Terminales: {CrearBD, EliminarBD, CrearColeccion, EliminarColeccion, InsertarUnico,

ActualizarUnico, EliminarUnico, BuscarUnico, BuscarTodo } No terminales: {,

Instrucciones>, }

Init: Instrucciones

Instrucciones: <Instrucción>

Instrucción:

CrearBD

EliminarBd

EliminarColeccion

InsertarUnico

ActualizarUnico

EliminarUnico

BuscarUnico

BuscarTodo

CrearBD :: CrearBD identificador = nueva CrearBD ( )

EliminarBD | EliminarBD identificador = nueva EliminarBD ( )

CrearColeccion | CrearColeccion identificador = nueva CrearColeccion ( " STRING " )

EliminarColeccion | EliminarColeccion identificador = nueva EliminarColeccion ( "

STRING " )

InsertarUnico | InsertarUnico identificador = nueva InsertarUnico ( " STRING " , "

STRINGJSON " )

ActualizarUnico | ActualizarUnico identificador = nueva ActualizarUnico ( " STRING

, " STRINGJSON " )

EliminarUnico | EliminarUnico identificador = nueva EliminarUnico ( " STRING " , "

STRINGJSON " )

BuscarUnico | BuscarUnico identificador = nueva BuscarUnico ( " STRING " )

BuscarTodo | BuscarTodo identificador = nueva BuscarTodo ( " STRING " )