MANUAL TECNICO

Analizador Léxico Application:

Tabla de Contenido

Introducción ...……..1
Objetivos ……...1
Información destacada ……...2
Conocimientos previos ……2
1. Requerimientos ……...3
2. Instalación y configuración ……4
3. Estructura del programa ……..……5
4. Paradigma utilizado …….……...……..…..6
5. Fragmentos de códigos ..……...…….6

Introducción

El presente documento describe la seria de pasos que realizan algunos de los métodos o funciones que conforman el programa computacional (AnalizadorLexicoApp) por medio de fragmentos de código, como también se detalla los conocimientos previos que debe tener el lector de este manual para comprender de una mejor manera el funcionamiento de cada una de las partes del código que conforman el programa.

Objetivos

Instruir el uso adecuado del programa computacional, describiendo el diseño y la lógica del programa por medio de fragmentos de códigos.

Describir al usuario el funcionamiento del programa para el mejor uso de él y demostrar el proceso necesario para su ejecución.

Orientar al usuario a entender la estructura del programa, como lo son sus clases y cada uno de los métodos que componen dicha clase.

Información destacada

El manual técnico hace referencia a la información necesaria con el fin de orientar al personal en la concepción, planteamiento análisis programación e instalación del sistema. Es de notar que la redacción propia del manual técnico está orientada a personal con conocimientos en sistemas y tecnologías de información, conocimientos de programación avanzada sobre Python, responsables del mantenimiento e instalación del programa computacional en el computador.

Conocimientos Previos

Los conocimientos mínimos que deben tener las personas que operarán las páginas y deberán utilizar este manual son:

- Cocimientos y entendimientos en logaritmos
- Conocimientos en Python
- Programación Orientada a Objetos
- Interfaces graficas (Tkinter)
- Conocimiento básico de Windows

1. Requerimientos

El sistema puede ser instalado en cualquier sistema operativo que cumpla con los siguientes requerimientos:

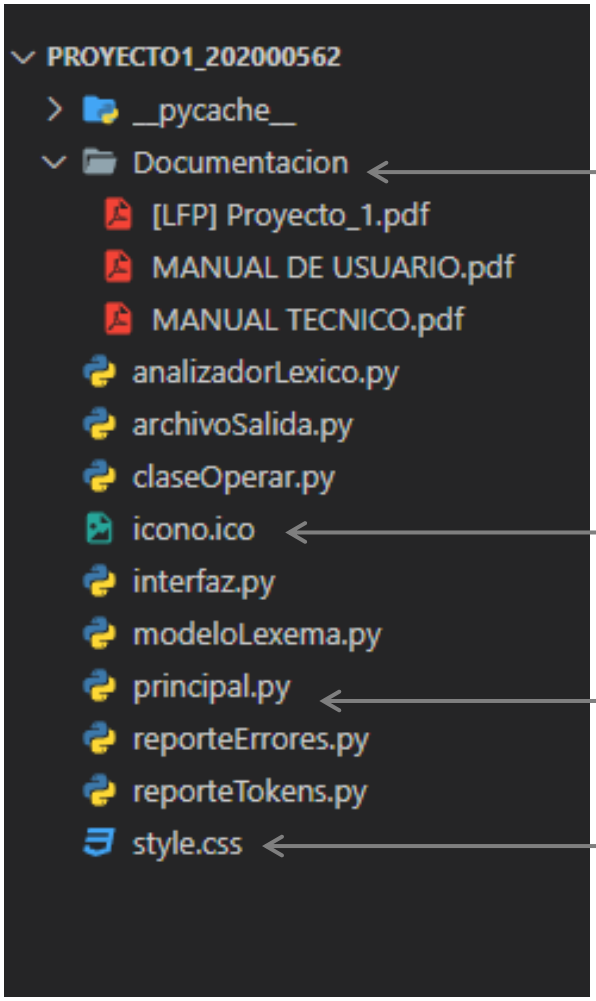
- Sistema Operativo: Windows 7 o superior
- Procesador mínimo Intel Pentium (800MHz Intel Pentium)
- Mínimo 1GB en RAM
- IDE Visual Studio Code, o compatible con Python
- Exploradores: Internet Explorer 9 y superior

2. Instalación y configuración

Para el proceso de instalación de esta aplicación únicamente es necesario tener instalado un IDE que sea compatible con el lenguaje Python para ejecutar la aplicación desde la terminal de este.

No es necesario tener alguna configuración ya que la configuración que trae por determinado el IDE es la necesaria para que el funcionamiento del programa sea posible.

3. Estructura del programa



Esta carpeta contiene toda la documentación necesaria para comprender de una mejor manera la estructura y el uso adecuado del programa.

Esta imagen es el icono de la aplicación.

Este es el archivo principal el cual se ejecuta al correr el programa.

Este archivo da el diseño a las tablas de errores y tokens del programa.

4. Paradigma de Programación Usado

Para el desarrollo de este programa se utilizo el paradigma de programación orientada a objetos ya que este disminuye los errores y promociona la reutilización del código.

El paradigma de la programación orientada a objetos consiste en la representación de la realidad. En éste se manejan algunos conceptos básicos como son clases, objetos, atributos, métodos y se caracteriza por emplear la abstracción de datos, herencia, encapsulamiento y polimorfismo.

5. Fragmentos de Códigos

En este apartado se explican detalladamente los método y funciones más importantes que conforman el código del programa. Esto con el objetivo de que la persona a usar el programa necesite dar soporte a la aplicación se le realice una manera más sencilla comprender la lógica del programa.

Código del método “abrir”:

```
def abrir(self):
    self.ruta = filedialog.askopenfilename(title="Abrir")
    if self.ruta != "":
        ruta, extension = os.path.splitext(self.ruta)
        if extension.lower() == ".lfp":
            self.areaTexto.delete(1.0, "end")
            archivo = open(self.ruta, 'r')
            self.contenidoArchivo = archivo.readlines()
            archivo.close()
            self.contenidoArchivo = "".join(self.contenidoArchivo)
            self.areaTexto.insert('insert', self.contenidoArchivo)
            self.contenidoAreaTexto = self.areaTexto.get(1.0, "end-1c")
        else:
            messagebox.showwarning("Advertencia", "¡La extension del archivo es incorrecta!")
```

Este método primero verifica que el usuario haya seleccionado un archivo, de ser así la ruta del archivo será almacenada en la variable “self.ruta” de caso contrario no realizara nada. Luego de almacenar la ruta del archivo procede a verificar que la extensión del archivo seleccionado sea la correcta (.lfp) de no ser la correcta se mostrara un mensaje de error en pantalla.

Si la extensión del archivo es la correcta procede a leer línea por línea el archivo almacenando toda la información en la variable “self.contenidoArchivo” para luego ser mostrado todo el contenido en la área de texto de la aplicación.

Código del método “guardar”:

```
def guardar(self):
    self.contenidoAreaTexto = self.areaTexto.get(1.0, "end-1c")
    if self.ruta != "":
        archivo = open(self.ruta, "w")
        archivo.write(self.contenidoAreaTexto)
        archivo.close()
        self.contenidoArchivo = self.contenidoAreaTexto
    else:
        self.guardarComo()
```

El método “guardar” funciona de la siguiente manera:

Este método es llamado cuando el usuario presiona el botón “guardar” que se muestra en la interfaz gráfica de la aplicación y funciona de la siguiente manera:

Guarda todo el contenido que tiene el área de texto de la aplicación en una variable, luego verifica si el contenido pertenece a un archivo abierto por medio de la aplicación y de ser así realiza los cambios que el usuario ingreso en el archivo. En caso de que no haya un archivo abierto el programa le preguntara al usuario si desea guardar el contenido en un archivo nuevo.

Código del método “leerArchivo”

```
def leerArchivo(self, texto):
    self.Contenido += '      <div class="Contenido">\n'
    Areatexto = ""
    titulo = ""
    etiquetaTipo = False
    etiquetaTexto = False
    etiquetaFuncion = False
    etiquetaEstilo = False
    for linea in texto.split('\n'):
        tokenNumero = ""
        for lexema in linea:
            if lexema.isalpha() or lexema == '\t':
                self.scanner += lexema
            if etiquetaTipo:
                if lexema.isdigit() or lexema == '.':
                    tokenNumero += lexema
            if etiquetaTexto or etiquetaFuncion or etiquetaEstilo:
                Areatexto += lexema
```

El método “leerArchivo” se encuentra localizado en la clase “archivoSalida”, este método se encargar de recorrer línea por línea y columna por columna del contenido almacenado en el área de texto de la aplicación, el cual puede ser obtenido por medio de un archivo o ya se que el usuario escriba el texto en el área de texto.

El programa verifica que en la línea y columna en la que se encuentra cumpla con la condición de que se le impone.

En la primera condición verifica si el carácter leído es una letra o una tabulación, de ser así el carácter se concatenara a la variable “self.scanner”, luego en la siguiente condición verifica que el carácter leído sea un numero y si se cumple el carácter será almacenado en la variable “tokenNumero” y por ultimo en la siguiente condición verifica si una de las variables inicializadas al principio de la función es verdadera, y si esto es cierto el carácter leído será almacenado en una variable.

Código del método “s1”:

```
def s1(self,lexema:str):
    #ESTADO S1
    if lexema.isalpha():
        self.estado = 1
        self.scanner += lexema
        self.numColumna += 1
    elif lexema.isdigit():
        self.estado = 1
        self.scanner += lexema
        self.numColumna += 1
    elif lexema == '.':
        self.estado = 1
        self.scanner += lexema
        self.numColumna += 1
    else:
        if self.scanner in self.tokensReservados:
            self.numToken += 1
            self.agregarToken(self.numToken, self.scanner, "token reservado {}".format(self.scanner), self.numLinea, self.numColumna)
        else:
            self.numToken += 1
            self.agregarToken(self.numToken, self.scanner, "contenido", self.numLinea, self.numColumna)
        self.estado = 0
        self.i -= 1
```

Este método se encuentra en la clase “analizadorLexico” y este se encarga de verificar el estado al que pertenece el lexema leído por el programa perteneciente al contenido del área de texto de la aplicación.

El lexema pasa por varios condicionales, en los primeros tres condicionales se verifica si es letra, numero o punto de ser así, se concatena a la variable “self.scanner”. Si no se cumple con los primeros tres condicionales, el lexema se agrega a la lista “self.tokensReservados”.

Código del método “tipo”

```
def tipo(self, tokenNumero):
    try:
        if self.scanner.strip('\t') == 'OperacionSUMA':
            self.lexemasOperacion.append('+')
            self.abrirParentesis()
        elif self.scanner.strip('\t') == 'OperacionRESTA':
            self.lexemasOperacion.append('-')
            self.abrirParentesis()
        elif self.scanner.strip('\t') == 'NumeroNumero':
            self.concatenarSigno = not(self.concatenarSigno)
            if self.concatenarSigno and len(self.lexemasOperacion) > 0:
                lexemaOperacion = self.lexemasOperacion.pop()
            else:
                lexemaOperacion = ""
            if lexemaOperacion in ['sen', 'cos', 'tan']:
                self.operacion += lexemaOperacion + tokenNumero
            elif lexemaOperacion == "1/":
                self.operacion += tokenNumero
            else:
                self.operacion += tokenNumero + lexemaOperacion
        elif self.scanner.strip('\t') == 'Operacion':
            if not(self.operacion[-1].isdigit() or self.operacion[-1] == ")"):
                self.operacion = self.operacion[:-1]
            if self.scanner.count('\t') > 0:
                self.cerrarParentesis()
            else:
                operar = Operar(self.operacion)
                resultado = operar.operar()
                self.operacion += ' = '+ str(resultado)
                self.Contenido += '        <p>'+self.operacion+'</p>\n'
                self.operacion = ""
                self.lexemasOperacion = []
                self.parentesisAbiertos = 0
        self.scanner = ""
    #except Exception as e:
    except:
        messagebox.showerror("Error", "¡Ocurrio un error al analizar el documento!")
```

Este método se encuentra localizado en la clase “archivoSalida” y funciona así, primero verifica si lo que contiene la variable “self.scanner” coincide con el texto a comparar, de ser verdadero a la variable “self.operacion” se le concatenara un signo de operación perteneciente al condicional.

Si el contenido de la variable “scanner” es igual a “Operación”, de ser necesario el programa añadirá a la variable “self.operacion” un paréntesis cerrado. Por ultimo si ningún condicional se cumple el programa realizara la operación que contiene almacenada la variable “self.operacion”.