UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE CIENCIAS Y SISTEMAS

LENGUAJES FORMALES Y DE PROGRAMACIÓN

PRIMER SEMESTRE 2024

Manual Técnico

Descripción General

El código proporcionado es un programa en Python que utiliza la biblioteca Tkinter para crear una interfaz gráfica de usuario (GUI). El programa permite cargar datos de estudiantes desde un archivo de texto, cargar calificaciones de los estudiantes desde otro archivo de texto, generar informes en formato HTML sobre los datos de los estudiantes y generar un informe de aprobación de estudiantes basado en sus calificaciones. Además, proporciona la funcionalidad para mostrar los tres mejores estudiantes con el promedio más alto.

Requisitos del Sistema

- Python 3.x instalado en el sistema.
- La biblioteca Tkinter incluida en la instalación de Python.

Instalación y Configuración

1. Asegúrese de tener Python 3.x instalado en su sistema.
2. No se requiere una configuración adicional para la biblioteca Tkinter, ya que es una biblioteca estándar de Python.
3. Asegúrese de tener los archivos de datos necesarios en el mismo directorio que el script principal, o modifique las rutas de archivo según sea necesario.

Manual Técnico del Código

Descripción General:

El código proporcionado es una aplicación de escritorio desarrollada en Python utilizando la biblioteca Tkinter para la interfaz gráfica. Esta aplicación permite cargar un archivo de texto, analizar su contenido y traducirlo a un documento HTML. Además, proporciona opciones para reiniciar la aplicación, salir y abrir los archivos generados en un navegador web.

Estructura del Código:

El código se divide en varias secciones:

1. **Importaciones de módulos:** Importa los módulos necesarios para el funcionamiento de la aplicación, incluyendo `Analyzer` de un módulo llamado `analyzer`, `tkinter` para la interfaz gráfica, `filedialog` para abrir archivos, y `webbrowser` para abrir archivos HTML en el navegador.
2. **Funciones:**
    - `load_file()`: Carga un archivo de texto, analiza su contenido y lo muestra en un widget de texto.
    - `traducir()`: Traduce el contenido del archivo de texto a HTML y abre los archivos generados en el navegador.
    - `abrir_archivos()`: Abre los archivos HTML generados en el navegador.
    - `reiniciar()`: Reinicia la aplicación, limpiando los datos y los widgets de texto.

```python
def load_file():
    global analyzer
    global text
    global filepath
    file_path = filedialog.askopenfilename(filetypes=[("Text Files", "*.txt")])
    if file_path:
        with open(file_path, "r", encoding="utf-8") as file:
            text = file.read()
            analyzer = Analyzer(text)
            analyzer.analyze()
            print("====================== Tokens ======================")
            for token in analyzer.tokens:
                print(token)
            print("==================== Errores ======================")
            for error in analyzer.errors:
                print(error)
            textbox.delete("1.0", tk.END)
            textbox.insert(tk.END, text)

def traducir():
    analyzer.getTokens()
    leer_documento(text)
    crear_html("html.html")
    generarTexto()
    abrir_archivos()


def abrir_archivos():
    webbrowser.open_new_tab("html.html")
    webbrowser.open_new_tab("tokens.html")

def reiniciar():
    global analyzer
    global text
    global filepath
    analyzer = None
    text = None
    filepath = None
    formatear()
    textbox.delete("1.0", tk.END)
    html_textbox.delete("1.0", tk.END)
```

3. **Interfaz Gráfica:**
   - Configuración de la ventana principal (`root`).
   - Definición de widgets como botones y widgets de texto.

```
root = tk.Tk()
root.title("Traductor HTML")
root.resizable(False,False)
root.config(cursor="hand2")
root.geometry("1200x720")
root.config(bg="SlateBlue1")
root.config(bd="30")
root.config(relief="groove")


html_textbox = tk.Text(root, height=30, width=50)
html_textbox.pack(side=tk.RIGHT)

textbox = tk.Text(root, height=30, width=50)
textbox.pack(side=tk.LEFT)



button2 = tk.Button(root, text="Traducir", command=traducir,font=("Arial", 15))

button3 = tk.Button(root, text="Salir", command=root.quit, font=("Arial", 15))
button3.pack(side=tk.BOTTOM)
button4 = tk.Button(root, text="Reiniciar", command=reiniciar, font=("Arial", 15))

button = tk.Button(root, text="Cargar archivo", command=load_file, font=("Arial", 15))

button4.pack()
button.pack()
button2.pack()
button3.pack()
textbox.pack()
html_textbox.pack()
root.mainloop()
```

4. **Clase `ElementoHTML`:**
   - Define una clase para representar elementos HTML con instrucciones y atributos.

```
def crear_html(html_salida):
    with open(html_salida, 'w', encoding='utf-8') as archivo:
        archivo.write('<!DOCTYPE html>\n<html>\n<head>\n')
        for elemento in estructura:
            if elemento.instruccion == 'title':
                archivo.write(str(elemento))
                archivo.write('\n')
        archivo.write('</head>\n<body>\n')
        for elemento in estructura:
            if elemento.instruccion != 'title':
                archivo.write(str(elemento))
                archivo.write('\n')
        archivo.write('</body>\n</html>\n')

    print(f"Se ha creado el archivo HTML '{html_salida}'")
```

5. **Funciones para Procesar Bloques:**
    - Funciones para procesar diferentes tipos de bloques del archivo de texto y convertirlos en elementos HTML.
6. **Función `leer_documento()`:**
    - Lee el contenido del archivo de texto y procesa los bloques utilizando las funciones definidas anteriormente.

```
def leer_documento(ruta_archivo):
    contenido = ruta_archivo

    bloques_procesados = {
        'Encabezado': procesar_bloque_encabezado,
        'Titulo': generar_titulo,
        'Fondo': procesar_bloque_fondo,
        'Parrafo': procesar_bloque_parrafo,
        'Texto': procesar_bloque_texto,
        'Codigo': procesar_bloque_codigo,
        'Negrita': procesar_bloque_negrita,
        'Subrayado': procesar_bloque_subrayado,
        'Tachado': procesar_bloque_tachado,
        'Cursiva': procesar_bloque_cursiva,
        'Salto': procesar_bloque_salto,
        'elemento': procesar_bloque_elemento,
    }



    for palabra_clave, funcion_procesar in bloques_procesados.items():
        indice_busqueda = 0
        while True:
            indice_palabra_clave = contenido.find(palabra_clave, indice_busqueda)
            if indice_palabra_clave != -1:
                inicio_bloque = contenido.find('{', indice_palabra_clave)
                fin_bloque = contenido.find('}', inicio_bloque)
                if inicio_bloque != -1 and fin_bloque != -1:
                    bloque = contenido[inicio_bloque + 1: fin_bloque]
                    estructura.append(funcion_procesar(bloque))
                    indice_busqueda = fin_bloque + 1
                else:
                    break
            else:
                break

    return estructura
```

7. **Diccionarios:**
    - Diccionarios para mapear atributos como alineaciones, tamaños de texto y colores a sus equivalentes en HTML.
8. **Funciones `crear_html()` y `formatear()`:**
    - `crear_html()`: Crea un archivo HTML con los elementos procesados.
    - `formatear()`: Limpia la estructura de datos después de generar el HTML.

```python
def reiniciar():
    global analyzer
    global text
    global filepath
    analyzer = None
    text = None
    filepath = None
    formatear()
    textbox.delete("1.0", tk.END)
    html_textbox.delete("1.0", tk.END)
```

Diagrama del AFD generado con graphviz: