Documentación del código

Proyecto administrador de archivos

Ivan Alexander Tobar Villota
Jean Carlos Murillo Martinez

Estructuras de datos

UNIVERSIDAD DE CALDAS

2023-2

Documentación del código

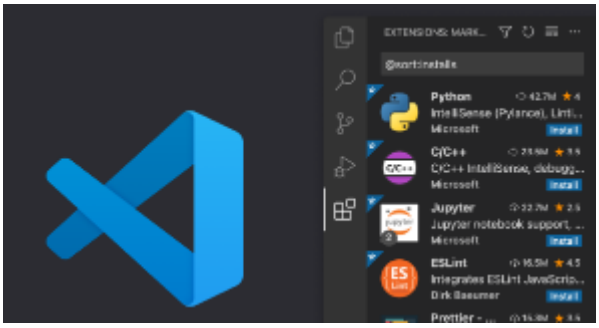
Lenguaje de programación

- **Python**

Python es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código, se utiliza para desarrollar aplicaciones de todo tipo.

Editor de código

- **Visual Studio Code**



Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web.

Módulos

- **Tkinter**

El paquete tkinter es una fina capa orientada a objetos encima de Tcl/Tk. Para usar tkinter , no necesita escribir código Tcl, pero deberá consultar la documentación de Tk y, ocasionalmente, la documentación de Tcl. tkinter es un conjunto de envoltorios que implementan los widgets Tk como clases de Python.

- **Módulo OS**

Este módulo provee una manera versátil de usar funcionalidades dependientes del sistema operativo.

Documentación del código

El código proporcionado es una implementación de diferentes funciones relacionadas con la manipulación de archivos y directorios en un sistema de archivos. Aquí está una breve documentación de las funciones más importantes:

```
def search_node(self):
    search_text = tk.simpledialog.askstring("Buscar Nodo", "Ingrese el nombre del nodo a buscar:")
    if not search_text:
        print("Ingrese un texto válido para la búsqueda.")
        return

    matching_nodes = self.find_matching_nodes(search_text)
    if not matching_nodes:
        print(f"No se encontraron nodos que coincidan con '{search_text}'.")
        return

    self.highlight_matching_nodes(matching_nodes)
```

- Muestra un cuadro de diálogo para ingresar el nombre del nodo a buscar.
- Si no se ingresa ningún texto, imprime un mensaje y sale de la función.

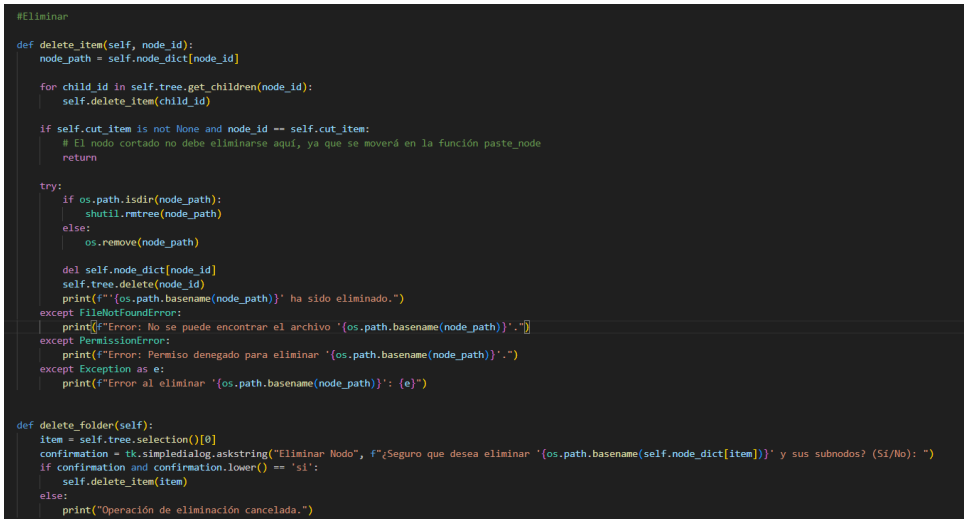
- Llama a find_matching_nodes para encontrar nodos que coincidan con el texto de búsqueda.
- Llama a highlight_matching_nodes para resaltar los nodos encontrados.

```python
def add_node(self):
    parent_item = self.tree.selection()[0]
    parent_path = self.node_dict[parent_item]

    new_node_name = tk.simpledialog.askstring("Agregar Nodo", "Ingrese el nombre del nuevo nodo:")
    new_node_path = os.path.join(parent_path, new_node_name)

    if not new_node_name:
        print("Ingrese un nombre válido para el nuevo nodo.")
        return

    if os.path.exists(new_node_path):
        print(f"Ya existe un nodo con el nombre '{new_node_name}'.")
        return

    if not os.path.exists(new_node_path):
        if new_node_name.lower().endswith(".txt"):
            with open(new_node_path, "w") as new_file:
                pass
        else:
            os.makedirs(new_node_path)

    new_node_id = self.tree.insert(parent_item, "end", text=new_node_name)
    self.node_dict[new_node_id] = new_node_path

    if os.path.isdir(new_node_path):
        self.expand_node(new_node_id)
```

```python
def expand_node(self, node_id):
    children = self.tree.get_children(node_id)
    if children:
        return
    node_path = self.node_dict[node_id]
    for child_path in os.listdir(node_path):
        child_full_path = os.path.join(node_path, child_path)
        if os.path.isdir(child_full_path):
            child_id = self.tree.insert(node_id, "end", text=os.path.basename(child_full_path))
            self.node_dict[child_id] = child_full_path
            self.expand_node(child_id)
        elif child_full_path.lower().endswith(".txt"):
            child_id = self.tree.insert(node_id, "end", text=os.path.basename(child_full_path))
            self.node_dict[child_id] = child_full_path

def OnDoubleClick(self, event):
    item = self.tree.selection()[0]
    node_path = self.node_dict[item]
    if os.path.isdir(node_path):
        self.expand_node(item)
```

Función add_node:
Obtiene el elemento seleccionado en el árbol (parent_item) para determinar el nodo padre.
Obtiene la ruta del nodo padre a partir del diccionario self.node_dict.
Muestra un cuadro de diálogo para ingresar el nombre del nuevo nodo.
Construye la ruta completa del nuevo nodo concatenando el nombre del nuevo nodo al path del nodo padre.
Verifica si se ingresó un nombre válido para el nuevo nodo y si ya existe un nodo con ese nombre.
Crea el nuevo nodo: si es un directorio, se crea un directorio utilizando os.makedirs, y si es un archivo de texto, se crea un archivo vacío utilizando open.
Inserta el nuevo nodo en el árbol y actualiza self.node_dict con la nueva información.

Función expand_node:
Expande un nodo en el árbol, es decir, muestra sus nodos hijos en la interfaz gráfica.
Obtiene la lista de nodos hijos del nodo actual.
Para cada ruta de nodo hijo en el sistema de archivos, inserta un nuevo nodo en el árbol y actualiza self.node_dict.
Llama recursivamente a expand_node para expandir los nodos hijos.
Función OnDoubleClick:
Maneja el evento de doble clic en un nodo del árbol.
Obtiene el elemento seleccionado en el árbol (item) y la ruta asociada al nodo desde self.node_dict.
Si el nodo es un directorio, expande el nodo llamando a la función expand_node.



```
#Eliminar

def delete_item(self, node_id):
    node_path = self.node_dict[node_id]

    for child_id in self.tree.get_children(node_id):
        self.delete_item(child_id)

    if self.cut_item is not None and node_id == self.cut_item:
        # El nodo cortado no debe eliminarse aquí, ya que se moverá en la función paste_node
        return

    try:
        if os.path.isdir(node_path):
            shutil.rmtree(node_path)
        else:
            os.remove(node_path)

        del self.node_dict[node_id]
        self.tree.delete(node_id)
        print(f"'{os.path.basename(node_path)}' ha sido eliminado.")
    except FileNotFoundError:
        print(f"Error: No se puede encontrar el archivo '{os.path.basename(node_path)}'.")
    except PermissionError:
        print(f"Error: Permiso denegado para eliminar '{os.path.basename(node_path)}'.")
    except Exception as e:
        print(f"Error al eliminar '{os.path.basename(node_path)}': {e}")


def delete_folder(self):
    item = self.tree.selection()[0]
    confirmation = tk.simpledialog.askstring("Eliminar Nodo", f"¿Seguro que desea eliminar '{os.path.basename(self.node_dict[item])}' y sus subnodos? (Sí/No): ")
    if confirmation and confirmation.lower() == 'sí':
        self.delete_item(item)
    else:
        print("Operación de eliminación cancelada.")
```

- **Función delete_item**:
- Toma un identificador de nodo (node_id) como argumento.
- Obtiene la ruta asociada al nodo desde self.node_dict.
- Recorre recursivamente y elimina todos los nodos hijos del nodo actual llamando a la función delete_item para cada hijo.
- Verifica si el nodo actual es el nodo que ha sido cortado (self.cut_item). Si es así, no lo elimina ya que se moverá más tarde en la función paste_node.
- Intenta eliminar el nodo actual en el sistema de archivos:
- Si es un directorio, utiliza shutil.rmtree para eliminar el directorio y su contenido.
- Si es un archivo, utiliza os.remove para eliminar el archivo.
- Elimina la entrada correspondiente del diccionario self.node_dict y el nodo del árbol.
- Imprime un mensaje indicando que el nodo ha sido eliminado.
- Captura posibles excepciones como FileNotFoundError (si el archivo no se encuentra), PermissionError (si no se tienen permisos para eliminar el archivo), y cualquier otra excepción general.
- **Función delete_folder**:
- Obtiene el elemento seleccionado en el árbol (item) para determinar qué nodo se eliminará.
- Muestra un cuadro de diálogo de confirmación preguntando al usuario si realmente desea eliminar el nodo y sus subnodos.
- Si el usuario confirma la eliminación, llama a la función delete_item con el identificador del nodo seleccionado.

- Si el usuario no confirma, imprime un mensaje indicando que la operación de eliminación ha sido cancelada.

```python
#Renombrar

def rename_node(self):
    item = self.tree.selection()[0]
    old_node_path = self.node_dict[item]
    new_node_name = tk.simpledialog.askstring("Renombrar Nodo", "Ingrese el nuevo nombre para el nodo:")
    new_node_path = os.path.join(os.path.dirname(old_node_path), new_node_name)

    if not new_node_name:
        print("Ingrese un nombre válido para el nuevo nodo.")
        return

    if os.path.exists(new_node_path):
        print(f"Ya existe un nodo con el nombre '{new_node_name}'.")
        return

    os.rename(old_node_path, new_node_path)

    self.node_dict[item] = new_node_path
    self.tree.item(item, text=new_node_name)
```

- **Obtención de información del nodo a renombrar**:
- Obtiene el elemento seleccionado en el árbol (item), que representa el nodo que se va a renombrar.
- Obtiene la ruta asociada al nodo desde el diccionario self.node_dict.
- **Solicitud de nuevo nombre al usuario**:
- Muestra un cuadro de diálogo para que el usuario ingrese el nuevo nombre para el nodo.
- Construye la nueva ruta completa (new_node_path) concatenando el directorio padre de la ruta antigua con el nuevo nombre proporcionado por el usuario.
- **Validación del nuevo nombre**:
- Verifica si se ingresó un nombre válido para el nuevo nodo. Si no, imprime un mensaje y sale de la función.
- **Verificación de duplicados**:
- Verifica si ya existe un nodo con el nuevo nombre en la ubicación especificada (new_node_path). Si es así, imprime un mensaje y sale de la función.
- **Renombrado del nodo en el sistema de archivos**:
- Utiliza os.rename para cambiar el nombre del nodo en el sistema de archivos de la ruta antigua (old_node_path) a la nueva ruta (new_node_path).
- **Actualización de la información en la interfaz gráfica y el diccionario**:
- Actualiza el diccionario self.node_dict con la nueva ruta del nodo.
- Actualiza el texto del nodo en el árbol con el nuevo nombre.

```python
## buscar

    def search_node(self):
        search_text = tk.simpledialog.askstring("Buscar Nodo", "Ingrese el nombre del nodo a buscar:")
        if not search_text:
            print("Ingrese un texto válido para la búsqueda.")
            return

        matching_nodes = self.find_matching_nodes(search_text)
        if not matching_nodes:
            print(f"No se encontraron nodos que coincidan con '{search_text}'.")
            return

        self.highlight_matching_nodes(matching_nodes)


    def find_matching_nodes(self, search_text, parent_node="", matching_nodes=None):
        if matching_nodes is None:
            matching_nodes = []

        for node_id, node_path in self.node_dict.items():
            if node_path.startswith(parent_node) and search_text.lower() in os.path.basename(node_path).lower():
                matching_nodes.append(node_id)

        print(f"Search Text: {search_text}, Matching Nodes: {matching_nodes}")
        return matching_nodes
```

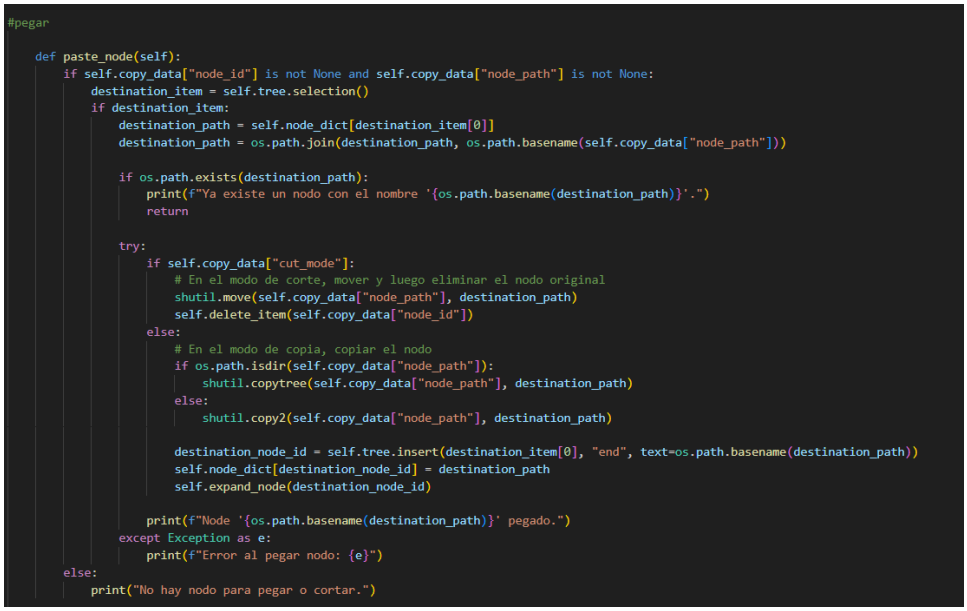
Este código es parte de un programa que utiliza la biblioteca Tkinter para crear una interfaz gráfica. En este fragmento, se implementa una función de búsqueda que permite al usuario ingresar un texto de búsqueda y luego busca y resalta nodos en la interfaz que coincidan con ese texto. La búsqueda se realiza en función de la ruta de los nodos y el texto proporcionado por el usuario.

```python
#copiar

    def copy_node(self):
        selected_item = self.tree.selection()
        if selected_item:
            self.copy_data["node_id"] = selected_item[0]
            self.copy_data["node_path"] = self.node_dict[selected_item[0]]
            self.copy_data["cut_mode"] = False
            print(f"Node '{os.path.basename(self.copy_data['node_path'])}' copiado.")
```

copy_node se encarga de guardar información sobre el nodo actualmente seleccionado en la interfaz gráfica en un diccionario llamado copy_data. Esta información incluye el ID del nodo, la ruta del nodo y un indicador de modo de operación (cut_mode), que se establece en False para indicar que el nodo se está copiando (no cortando). Además, la función imprime un mensaje en la consola indicando que el nodo ha sido copiado, junto con el nombre del nodo. Este código es parte de un programa que probablemente maneja operaciones de copia y pegado en una estructura de árbol de nodos.

```python
#cortar

    def cut_node(self):
        selected_item = self.tree.selection()
        if selected_item:
            self.copy_data["node_id"] = selected_item[0]
            self.copy_data["node_path"] = self.node_dict[selected_item[0]]
            self.copy_data["cut_mode"] = True
            print(f"Node '{os.path.basename(self.copy_data['node_path'])}' cortado.")
```

la función cut_node se encarga de guardar información sobre el nodo actualmente seleccionado en la interfaz gráfica en un diccionario llamado copy_data. Esta información incluye el ID del nodo, la ruta del nodo y un indicador de modo de operación (cut_mode), que se establece en True para indicar que el nodo se está cortando. Además, la función imprime un mensaje en la consola indicando que el nodo ha sido cortado, junto con el nombre del nodo. Este código es parte de un programa que probablemente maneja operaciones de corte y pegado en una estructura de árbol de nodos.



```
#pegar

    def paste_node(self):
        if self.copy_data["node_id"] is not None and self.copy_data["node_path"] is not None:
            destination_item = self.tree.selection()
            if destination_item:
                destination_path = self.node_dict[destination_item[0]]
                destination_path = os.path.join(destination_path, os.path.basename(self.copy_data["node_path"]))

                if os.path.exists(destination_path):
                    print(f"Ya existe un nodo con el nombre '{os.path.basename(destination_path)}'.")
                    return

                try:
                    if self.copy_data["cut_mode"]:
                        # En el modo de corte, mover y luego eliminar el nodo original
                        shutil.move(self.copy_data["node_path"], destination_path)
                        self.delete_item(self.copy_data["node_id"])
                    else:
                        # En el modo de copia, copiar el nodo
                        if os.path.isdir(self.copy_data["node_path"]):
                            shutil.copytree(self.copy_data["node_path"], destination_path)
                        else:
                            shutil.copy2(self.copy_data["node_path"], destination_path)

                        destination_node_id = self.tree.insert(destination_item[0], "end", text=os.path.basename(destination_path))
                        self.node_dict[destination_node_id] = destination_path
                        self.expand_node(destination_node_id)

                    print(f"Node '{os.path.basename(destination_path)}' pegado.")
                except Exception as e:
                    print(f"Error al pegar nodo: {e}")
        else:
            print("No hay nodo para pegar o cortar.")
```

la función maneja el proceso de pegado, ya sea moviendo (en el caso de corte) o copiando (en el caso de copia) un nodo desde el portapapeles a una ubicación de destino en la estructura del árbol de nodos. Se considera la existencia de un nodo en el portapapeles y la selección de un destino en el árbol.