

Claro, aquí tienes una explicación detallada línea por línea del código Python que has proporcionado.

El script `RenombrarFichero.py` crea una aplicación de escritorio con una interfaz gráfica (GUI) para renombrar archivos en un directorio de forma masiva. Permite añadir prefijos, sufijos, reemplazar texto o añadir una numeración secuencial a los nombres de los archivos.

Análisis del Código

Aquí tienes el desglose de lo que hace cada parte del script:

1. Importaciones

Python

```
import sys
import os
from PyQt6.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QPushButton, QFileDialog,
    QLabel, QListWidget, QLineEdit, QHBoxLayout, QRadioButton, QGroupBox,
    QMessageBox
)
from PyQt6.QtCore import Qt
```

- `import sys`: Se importa para interactuar con el sistema, principalmente para gestionar los argumentos de la línea de comandos al iniciar la aplicación y para cerrarla correctamente con `sys.exit()`.
- `import os`: Esencial para interactuar con el sistema operativo. Se usa para listar archivos de un directorio (`os.listdir`), comprobar si una ruta es un archivo (`os.path.isfile`), unir rutas (`os.path.join`), y renombrar archivos (`os.rename`).
- `from PyQt6.QtWidgets import ...`: Importa todos los componentes gráficos (widgets) necesarios de la librería PyQt6 para construir la interfaz de usuario.
 - `QApplication`: Gestiona el bucle de eventos principal de la aplicación.
 - `QWidget`: Es la clase base para todos los objetos de la interfaz de usuario. La ventana principal heredarán de ella.
 - `QVBoxLayout`, `QHBoxLayout`: Clases para organizar los widgets verticalmente y horizontalmente.
 - `QPushButton`: Un botón estándar.
 - `QFileDialog`: Un diálogo para que el usuario seleccione archivos o directorios.

- QLabel: Una etiqueta para mostrar texto.
- QListWidget: Un widget para mostrar una lista de elementos (en este caso, los nombres de los archivos).
- QLineEdit: Un campo de entrada de texto de una sola línea.
- QRadioButton: Un botón de opción que permite seleccionar una de varias opciones excluyentes.
- QGroupBox: Un contenedor con un borde y un título para agrupar widgets.
- QMessageBox: Un cuadro de diálogo para mostrar mensajes informativos o de confirmación.
- from PyQt6.QtCore import Qt: Se importa para acceder a ciertas constantes y enumeraciones de Qt, aunque en este código específico no se llega a usar explícitamente una de sus constantes más comunes como las de alineación.

2. Definición de la Clase Principal

Python

```
class FileRenamerApp(QWidget):
    def __init__(self):
        super().__init__()
        self.current_directory = ""
        self.init_ui()
```

- class FileRenamerApp(QWidget):: Define la clase principal de la aplicación, que hereda de QWidget. Esto significa que FileRenamerApp es un widget y actuará como la ventana principal.
- def __init__(self):: El constructor de la clase.
- super().__init__(): Llama al constructor de la clase padre (QWidget) para inicializarla correctamente.
- self.current_directory = "": Inicializa una variable de instancia para almacenar la ruta del directorio seleccionado por el usuario. Al principio está vacía.
- self.init_ui(): Llama al método que se encargará de crear y organizar todos los componentes de la interfaz gráfica.

3. Inicialización de la Interfaz de Usuario (init_ui)

Este método construye la apariencia de la ventana.

Python

```
def init_ui(self):
    self.setWindowTitle("Renombrador de Ficheros")
    self.setGeometry(100, 100, 800, 600)
```

```
main_layout = QVBoxLayout()
```

- `self.setWindowTitle(...)`: Establece el título de la ventana.
- `self.setGeometry(...)`: Define la posición y el tamaño inicial de la ventana en la pantalla (x, y, ancho, alto).
- `main_layout = QVBoxLayout()`: Crea un layout vertical que será el contenedor principal de todos los demás widgets y layouts.

Python

```
# --- Selección de Directorio ---
dir_selection_layout = QHBoxLayout()
self.dir_label = QLabel("Directorio Seleccionado: Ninguno")
dir_selection_layout.addWidget(self.dir_label)
self.select_dir_button = QPushButton("Seleccionar Directorio")
self.select_dir_button.clicked.connect(self.select_directory)
dir_selection_layout.addWidget(self.select_dir_button)
main_layout.addLayout(dir_selection_layout)
```

- Crea un layout horizontal (QHBoxLayout) para la sección de selección de directorio.
- `self.dir_label`: Una etiqueta que mostrará la ruta del directorio elegido.
- `self.select_dir_button`: Un botón que, al ser pulsado (clicked), ejecutará el método `self.select_directory`.
- `main_layout.addLayout(...)`: Añade este layout horizontal al layout principal vertical.

Python

```
# --- Lista de Ficheros ---
self.file_list_widget = QListWidget()
main_layout.addWidget(self.file_list_widget)
```

- `self.file_list_widget`: Crea el widget que mostrará la lista de archivos del directorio seleccionado o la previsualización de los cambios.
- `main_layout.addWidget(...)`: Lo añade al layout principal.

Python

```
# --- Opciones de Renombrado ---
options_group_box = QGroupBox("Opciones de Renombrado")
# ... (creación de RadioButtons y layouts para las opciones) ...
main_layout.addWidget(options_group_box)
```

- Esta sección crea un QGroupBox para agrupar visualmente todas las opciones de renombrado.
- Dentro de este grupo, se crean los cuatro QRadioButton para cada una de las acciones posibles (añadir prefijo, sufijo, reemplazar, numerar).
- Se conectan los botones de radio al método self.toggle_options_visibility para que la interfaz se actualice dinámicamente.
- También se crean los campos de texto (QLineEdit) necesarios para cada opción (por ejemplo, el campo para escribir el prefijo).

Python

```
self.toggle_options_visibility() # Set initial visibility
```

- Llama a este método una vez al inicio para asegurarse de que solo los campos de la opción seleccionada por defecto (o ninguna) sean visibles.

Python

```
# --- Botones de Acción ---
self.preview_button = QPushButton("Previsualizar")
self.preview_button.clicked.connect(self.preview_changes)

self.apply_button = QPushButton("Aplicar Cambios")
self.apply_button.clicked.connect(self.apply_changes)
```

- Crea los botones "Previsualizar" y "Aplicar Cambios" y los conecta a sus respectivos métodos: self.preview_changes y self.apply_changes.

Python

```
self.setLayout(main_layout)
```

- Finalmente, establece el `main_layout` como el layout principal de la ventana `FileRenamerApp`.

4. Métodos de Funcionalidad

Python

```
def select_directory(self):
    directory = QFileDialog.getExistingDirectory(self, "Seleccionar Directorio")
    if directory:
        self.current_directory = directory
        self.dir_label.setText(f"Directorio Seleccionado: {self.current_directory}")
        self.load_files()
```

- `QFileDialog.getExistingDirectory(...)`: Abre un diálogo estándar del sistema para que el usuario elija una carpeta.
- `if directory::` Si el usuario selecciona una carpeta y no cancela, el código dentro del `if` se ejecuta.
- Actualiza `self.current_directory` con la ruta elegida.
- Actualiza el texto de la etiqueta `self.dir_label`.
- Llama a `self.load_files()` para cargar y mostrar los archivos de ese directorio.

Python

```
def load_files(self):
    self.file_list_widget.clear()
    if self.current_directory:
        files = [f for f in os.listdir(self.current_directory) if
os.path.isfile(os.path.join(self.current_directory, f))]
        files.sort()
        for f in files:
            self.file_list_widget.addItem(f)
```

- `self.file_list_widget.clear()`: Limpia la lista de archivos antes de cargar nuevos.
- `os.listdir(...)`: Obtiene una lista con todos los nombres de archivos y carpetas en el directorio.
- `[f for f in ... if os.path.isfile(...)]`: Una "list comprehension" que filtra la lista para quedarse solo con los archivos (excluyendo las carpetas).
- `files.sort()`: Ordena los nombres de los archivos alfabéticamente.

- `self.file_list_widget.addItem(f)`: Añade cada nombre de archivo como un nuevo ítem a la lista visible en la interfaz.

Python

```
def toggle_options_visibility(self):
    # ... código para mostrar/ocultar los campos de texto ...
```

- Este método se activa cada vez que se cambia la selección de un `QRadioButton`.
- Su función es mostrar (`setVisible(True)`) únicamente los `QLineEdit` (campos de texto) que corresponden a la opción de renombrado seleccionada y ocultar (`setVisible(False)`) todos los demás. Esto hace que la interfaz sea más limpia y fácil de usar.

Python

```
def get_renamed_files(self):
    # ... lógica de renombrado ...
    return renamed_files_map
```

- Este es el cerebro de la lógica de renombrado. No modifica los archivos, solo calcula cómo se llamarían.
- Crea un diccionario `renamed_files_map` que almacenará { "nombre_antiguo": "nombre_nuevo" }.
- Dependiendo del `QRadioButton` que esté seleccionado:
 - **Añadir prefijo**: Concatena el texto del `prefix_input` al inicio del nombre del archivo.
 - **Añadir sufijo**: Concatena el texto del `suffix_input` al final del nombre, pero antes de la extensión.
 - **Reemplazar**: Usa el método `.replace()` de las cadenas para sustituir un texto por otro.
 - **Numeración automática**: Itera sobre los archivos y añade un número secuencial con ceros a la izquierda (ej. 001_, 002_).
- Devuelve el diccionario con los cambios propuestos.

Python

```
def preview_changes(self):
    self.file_list_widget.clear()
    renamed_files = self.get_renamed_files()
```

```

if renamed_files:
    # ... muestra "nombre_antiguo -> nombre_nuevo" en la lista
else:
    self.load_files()

```

- Llama a `get_renamed_files()` para obtener el mapa de cambios.
- Limpia la lista y la llena con cadenas formateadas como "archivo_viejo.txt -> archivo_nuevo.txt" para que el usuario vea el resultado antes de aplicarlo.
- Si no hay cambios para previsualizar, simplemente vuelve a cargar la lista de archivos original.

Python

```

def apply_changes(self):
    renamed_files = self.get_renamed_files()
    # ...
    for old_name, new_name in renamed_files.items():
        # ...
        os.rename(old_path, new_path)
        # ...
    self.load_files()
    QMessageBox.information(self, "Confirmación", "¡Cambios aplicados correctamente!")

```

- Este método es el que finalmente modifica los archivos en el disco.
- Obtiene el mapa de cambios de `get_renamed_files()`.
- Muestra un mensaje si no hay nada que hacer.
- Itera sobre el diccionario de cambios y usa `os.rename(ruta_antigua, ruta_nueva)` para efectuar el renombrado de cada archivo.
- Está envuelto en un `try...except` para capturar posibles errores (por ejemplo, si un archivo está en uso).
- Después de renombrar, llama a `self.load_files()` para refrescar la lista y mostrar los nuevos nombres.
- Muestra un `QMessageBox` para informar al usuario que la operación se ha completado.

5. Bloque de Ejecución Principal

Python

```

if __name__ == "__main__":

```

```
app = QApplication(sys.argv)
window = FileRenamerApp()
window.show()
sys.exit(app.exec())
```

- `if __name__ == "__main__":`: Este es un bloque estándar en Python que asegura que el código que contiene solo se ejecute cuando el script se corre directamente (y no cuando se importa desde otro script).
- `app = QApplication(sys.argv)`: Crea la instancia de la aplicación.
- `window = FileRenamerApp()`: Crea una instancia de nuestra clase principal.
- `window.show()`: Hace visible la ventana en la pantalla.
- `sys.exit(app.exec())`: Inicia el bucle de eventos de la aplicación. La aplicación esperará aquí a que el usuario interactúe con ella (haga clics, escriba, etc.) y no terminará hasta que se cierre la ventana. `sys.exit()` asegura una salida limpia.