**Documentación del archivo controlador.py**

**Propósito**

El archivo controlador.py contiene la clase Controlador, que es responsable de gestionar las interacciones entre la interfaz de usuario (UI) y los diferentes módulos del programa (cálculo, gráficos, almacenamiento de datos). La principal función del controlador es recibir las solicitudes del usuario y delegar las tareas a los módulos correspondientes, gestionando las excepciones y manteniendo la continuidad del programa incluso si un módulo falla al importarse.

**Estructura del Código**

1. **Importaciones Condicionales**: Cada módulo (cálculo, gráficos, almacenamiento y UI) es importado dentro de un bloque try-except. Si la importación de un módulo falla, el controlador lo maneja asignando None a la variable que representa ese módulo y mostrando un mensaje de error.
   * **realizar\_calculo**: Módulo para realizar cálculos matemáticos y científicos.
   * **generar\_grafico**: Módulo para generar gráficos.
   * **guardar\_datos**: Módulo para guardar los datos en archivos locales.
   * **InterfazUsuario**: Clase para gestionar la interfaz gráfica de usuario.
2. **Clase Controlador**:
   * La clase Controlador actúa como el cerebro del programa, gestionando las solicitudes del usuario y enviándolas a los módulos correspondientes.

**Métodos Principales**

**1. \_\_init\_\_(self)**

* **Descripción**: Este es el constructor de la clase Controlador. Se encarga de inicializar la interfaz de usuario y la muestra al inicio del programa.
* **Función**: Crea una instancia de la clase InterfazUsuario y la pasa al controlador para su manejo. Si la importación del módulo de la UI falla, el controlador muestra un mensaje de error y no crea la interfaz gráfica.

self.ui = InterfazUsuario(self) # Inicializa la UI

self.ui.mostrar() # Muestra la UI si está disponible

**2. ejecutar\_accion(self, accion, \*\*kwargs)**

* **Descripción**: Este método recibe las acciones del usuario (como "calcular", "graficar" o "guardar"), valida qué módulo debe ejecutar la acción y pasa los datos necesarios a ese módulo.
* **Parámetros**:
  + **accion** (str): El tipo de acción que el usuario desea realizar (por ejemplo, 'calculo', 'grafico', 'guardar').
  + **\*\*kwargs**: Argumentos adicionales necesarios para realizar la acción (por ejemplo, valores para un cálculo o datos para graficar).
* **Función**: Dependiendo del tipo de acción, el controlador invoca el módulo correspondiente:
  + Si la acción es 'calculo', llama al módulo de cálculos (realizar\_calculo).
  + Si la acción es 'grafico', llama al módulo de gráficos (generar\_grafico).
  + Si la acción es 'guardar', llama al módulo de almacenamiento de datos (guardar\_datos).
  + Si un módulo no está disponible (no se pudo importar), el controlador muestra un mensaje de error.

if accion == 'calculo' and realizar\_calculo:

resultado = realizar\_calculo(tipo\_calculo, valores)

self.ui.mostrar\_resultado(resultado)

elif accion == 'grafico' and generar\_grafico:

generar\_grafico(datos)

self.ui.mostrar\_mensaje("Gráfico generado.")

elif accion == 'guardar' and guardar\_datos:

guardar\_datos(nombre\_archivo, datos)

self.ui.mostrar\_mensaje(f"Datos guardados en {nombre\_archivo}")

else:

self.ui.mostrar\_mensaje("Acción no soportada o módulo no disponible.")

**3. Manejo de Errores de Importación**

* **Descripción**: Si uno de los módulos no se puede importar (por ejemplo, si el archivo del módulo no está presente o hay un error en la instalación), el controlador maneja la excepción, imprime un mensaje de error en la consola y asigna None a la variable que representa el módulo.
* **Objetivo**: Asegurarse de que el programa no se detenga por completo si un módulo falla, y continuar ejecutando otras funcionalidades que no dependan del módulo problemático.
* **Ejemplo**: Si el módulo de gráficos no se puede importar:
* except ImportError as e:
* print("Error al importar el módulo de gráficos:", e)
* generar\_grafico = None # Si falla la importación, asignamos None

**Flujo de Ejecución**

1. **Inicialización**:
   * Cuando el programa se ejecuta, se crea una instancia de la clase Controlador, lo que inicia el proceso de importación de módulos.
   * Si la interfaz de usuario se importa correctamente, el controlador la muestra al usuario.
2. **Acciones del Usuario**:
   * La interfaz de usuario envía una solicitud al controlador para realizar una acción específica, como calcular un valor, generar un gráfico o guardar datos.
   * El controlador maneja la acción, llama al módulo correspondiente y muestra el resultado en la UI.
3. **Manejo de Módulos No Disponibles**:
   * Si algún módulo no se puede importar, el controlador maneja el error y muestra un mensaje de error en la consola. Las acciones que no dependan de ese módulo seguirán funcionando correctamente.

**Consideraciones**

* **Modularidad**: El controlador permite una fácil extensión del programa. Si más adelante deseas agregar nuevos módulos (como un módulo de "análisis de datos" o "estadísticas"), solo tienes que importarlos en el controlador y gestionar las acciones correspondientes.
* **Robustez**: El manejo de errores en la importación de módulos asegura que el programa siga funcionando incluso si algunos módulos no están disponibles. Esto mejora la experiencia del usuario, ya que no se interrumpe la ejecución del programa.
* **Escalabilidad**: A medida que agregues más funcionalidades o módulos, puedes seguir ampliando el método ejecutar\_accion para incluir nuevas acciones sin afectar el resto del programa.

**Conclusión**

El archivo controlador.py es el núcleo del programa, coordinando las interacciones entre la interfaz de usuario y los módulos de funcionalidad. Gracias al manejo de excepciones y el modularidad, el programa puede continuar funcionando incluso si algunos módulos no están disponibles, mejorando la estabilidad y robustez del software.