INFORME DE FUNCIONALIDAD Y DISEÑO

En el código se pueden identificar varios principios y criterios de diseño. Aquí hay algunos de ellos:

Separación de Responsabilidades:

Organicé el código en clases donde cada una tiene responsabilidades específicas. La clase **`ticker_base_datos`** se encarga de la manipulación de la base de datos, la clase **`ticker_actualizar`** maneja la actualización de datos desde la API, la clase **`validacion_datos`** se encarga de validar la entrada del usuario, y la clase **`sub_menus`** se ocupa de las interacciones de visualización y menús. Más adelante se detalla cada clase y sus métodos.

Uso de Clases:

En el código utilizo clases para organizar funcionalidades relacionadas. Esto me facilita la encapsulación y la gestión de estado.

Manejo de Excepciones:

En el código incorporé bloques **`try`** y **`except`** para manejar excepciones, lo que me ayuda a anticipar y manejar posibles errores durante la ejecución.

Uso de Funciones y Métodos:

La lógica del programa la he dividido en funciones y métodos, lo que me facilita la lectura, la reutilización y el mantenimiento del código.

Integración con API Externa:

La clase **`ticker_actualizar`** se encarga de gestionar la interacción con una API externa para la obtención de datos de ticker. Este enfoque demuestra la implementación de un diseño estructurado al encapsular la lógica de la API dentro de una clase específica.

Manejo de Gráficos:

He implementado una función para graficar datos, lo que agrega una capa de visualización a la aplicación.

Manejo de Conexiones a la Base de Datos:

El código maneja las conexiones a la base de datos, pero podría beneficiarse de un manejo más robusto, si lograba usar los bloques **`with`** para garantizar el cierre adecuado de las conexiones. Lo intenté, pero no tuve éxito.

Menús Interactivos:

Implementé menús interactivos que permiten al usuario seleccionar opciones y realizar diferentes acciones en función de esas elecciones, incluso añadí la funcionalidad que desde un menú pueda acceder al anterior y que cuando termine la operación decida entre salir de la aplicación o volver al menú principal.

Cierre de la Conexión de Base de Datos:

La conexión de la base de datos se cierra al final del programa, lo cual es una buena práctica para asegurarse de que no se dejen conexiones abiertas.

En general, el código muestra esfuerzos para seguir principios de diseño orientado a objetos y buenas prácticas de programación. Sin embargo, hay áreas donde se que podría mejorar la claridad. A continuación, se describen las principales funcionalidades y componentes del código:

1. Clase `ticker_base_datos`:

Inicialización **`(__init__)`**: La clase inicia una conexión a una base de datos SQLite llamada 'TickerBaseDatos.db' y crea una tabla principal llamada 'TickerGuardados' para almacenar información sobre los tickers, incluyendo el nombre del ticker, la fecha de inicio y la fecha final de los datos asociados.

1.1. Métodos para manipular la tabla principal:

- ✓ **`tabla_principal_crear`**: Crea la tabla 'TickerGuardados' si no existe.
- ✓ **`tabla_principal_insertar`**: Inserta registros en la tabla 'TickerGuardados'.
- ✓ **`tabla_principal_actualizar`**: Actualiza los valores de fecha en la tabla 'TickerGuardados'.
- ✓ **`tabla_principal_ordenar`**: Crea una tabla ordenada y reemplaza la original para mantener la clasificación por nombre del ticker.
- ✓ **`tabla_principal_buscar`**: Recupera la lista de tickers guardados y devuelve un DataFrame con la información de la tabla principal.
- ✓ **`tabla_principal_visualizar`**: Imprime en consola la información de la tabla principal, los tickers almacenados y las fechas correspondientes.

1.2. Métodos para manipular tablas de tickers individuales:

- ✓ **`tabla_ticker_crear`**: Crea una tabla específica para un ticker y la llena con datos provenientes de la API.
- ✓ **`tabla_ticker_insertar`**: Inserta datos en la tabla específica de un ticker.
- ✓ **`tabla_ticker_ordenar`**: Ordena la tabla específica de un ticker por fecha.
- ✓ **`tabla_ticker_borrar`**: Elimina la tabla de un ticker en específico.
- ✓ **`tabla_ticker_buscar`**: Recupera la información de la tabla específica de un ticker.

1.3. Método `cerrar_bd`: Cierra la conexión con la base de datos.

2. Clase `ticker_actualizar`:

Inicialización **`(__init__)`**: Recibe una instancia de la clase **`ticker_base_datos`** para realizar operaciones de actualización.

Métodos:

- ✓ **`solicitar_datos_ticker`**: Realiza una solicitud a la API para obtener datos de un ticker en un rango de fechas. En este método se realiza el manejo de errores y reconexión a la API.
  Incluso se considera la respuesta del requests cuando no tenemos autorización para solicitar datos de un cierto rango temporal.

- ✓ **`validar_ticker`**: Verifica la existencia de un ticker mediante una solicitud a la API.

- ✓ **`verificacion_datos`**: Verifica y actualiza los datos de un ticker en la base de datos. Este método realiza la comparación con los datos que ya tenemos almacenados en la base de datos, si el ticker solicitado no existe en nuestra base de datos, creará la tabla, pero si el ticker existe, entonces comparará las fechas de la solicitud ingresadas y las fechas de los datos almacenados antes de hacer un hacer un request a la API. Esto fue solicitado como un extra.

3. Clase `validacion_datos`:

Inicialización **`(__init__)`**: Recibe instancias de **`ticker_actualizar`** y **`ticker_base_datos`** para realizar validaciones.

Métodos:

- ✓ **`validar_ticker`**: Solicita al usuario un ticker y primero valida su existencia en nuestra base de base de datos y si no la encuentra, la validará haciendo un request a la API.

- ✓ **`validar_fechas`**: Solicita al usuario fechas y realiza validaciones. Solicita que la fecha sea ingresada en el formato correcto, y luego verifica que la fecha inicial sea menor a la fecha final, y que ambas sean menores a la fecha actual.

4. Clase `sub_menus`:

Inicialización **`(__init__)`**: Recibe una instancia de **`ticker_base_datos`** para realizar operaciones de visualización.

Métodos:

- ✓ **`visualizar`**: Presenta un menú para la visualización de datos, incluyendo resumen y gráfico de un ticker.

- ✓ **`graficarTicker`**: Muestra gráficos de datos de un ticker específico. El ticker tiene que estar guardado en nuestra base de datos, de lo contrario, te pedirá que lo solicites desde el menú principal. Luego te muestra los parámetros disponibles para imprimir.

- ✓ **`consultaFinal`**: Pregunta al usuario si desea regresar al menú principal o salir.

5. Función `main()`:

Inicia las instancias necesarias y presenta un menú principal con opciones para actualizar, visualizar datos o salir del programa.