Reflections on the Decision Process and Design Choices:

Choice of Python and Use of NBP APIs.

Decision: Python was chosen because of its versatility, rich data processing library (e.g., pandas), and ease of integration with the API.

Reflection: Python is a popular language for data analysis, making it an ideal choice for financial data processing and analysis tasks. The use of the NBP API provides access to up-to-date currency data, which is crucial to the script's effectiveness.

Structure and Organization of the Code

Decision: The code was divided into functions corresponding to each task (data retrieval, data recording, analysis, etc.).

Reflection: The division into functions makes the code more readable and easier to maintain. It also allows the script to be easily extended and modified in the future.

Error Handling

Decision: Error handling has been implemented for various scenarios, such as API connection errors or file saving issues.

Reflection: Error handling is crucial to the reliability of the script. Informing the user about errors in a friendly way improves usability and allows for faster troubleshooting.

User Interaction

Decision: The user is able to select currency pairs to analyze and write, and receives confirmation after the operation is performed.

Reflection: Interactivity improves the flexibility of the script and its suitability for different users with different needs.

Automation and Task Scheduling

Decision: Implemented automatic running of the script every day at 12:00.

Reflection: Automation provides regular and up-to-date data without having to run the script manually, which is very convenient for users.

Data Testing and Validation

Decision: Simple input data validation and API response testing has been implemented.

Reflection: Testing and validation are essential to ensure data reliability and accuracy. For a script that operates on financial data, data accuracy and validation are especially important.

Documentation and Clean Code

Decision: Comments and documentation of functions (docstrings) were added to explain the operation of different parts of the code.

Reflection: Good documentation is crucial for understanding and maintainability of the code, especially when more than one person is working on it or when the code will be reviewed after a long time.

Consideration of Future Extensions

Decision: The code has been written in a way that makes it easy to add new functionality, such as additional currency pairs or other forms of analysis.

Reflection: Flexibility in design allows the script to easily adapt to changing requirements or user needs.

Balancing Complexity with Usability

Decision: The script is designed to be relatively simple to use, but at the same time provide enough functionality to perform the needed analysis.

Reflection: It is important to strike a balance between complexity and intuitiveness of use. A script that is too complex may deter less experienced users, while one that is too simplistic may not meet all the needs of more advanced users.

Conclusion

In designing the script, it was important to understand the user's needs and ensure that the script is not only functional, but also easy to use and adaptable to different scenarios. In the future, there may be a need for further improvements, such as better data validation or expanded data analysis functionality. Designing with future extensions and enhancements in mind is crucial to the long-term value and usability of the script.