Brandon Raupe
5/29/2023
Foundations of Programming: Python
Assignment 07
https://github.com/braupe1/-IntroToProg-Python-Mod07

Python Script: Pickle Binary Files with Functions and Try-Excepts

The Problem

The purpose of Assignment07 is to formulate the saving and processing of a binary file while importing the pickle library. This exercise intends to build upon previous lessons' skills with the aforementioned new twist.

The Script

This assignment reviews a Python script designed for managing a list of investment data (date and investment value) in a binary file. The script (found in Github at the above link) employs standard Python modules: `pickle` for serialization and deserialization and `datetime` for date manipulation. Furthermore, it uses a processor class to encapsulate the data manipulation functions and separates the presentation logic (I/O operations) from the processing logic, adhering to good software engineering practices.

The code script begins with an import statement, bringing in the `pickle` and `datetime` modules. `pickle` is used for serializing and deserializing Python objects for saving to or reading from a binary file. The `datetime` module, particularly the `datetime.strptime()` function, is used to validate and manipulate date entries.

The script establishes a filename (AppData.dat) for the binary file storing the investment data. This fixed assignment could be replaced with a more flexible approach, allowing users to specify the file name dynamically or read it from a configuration file.

The script structures its processing functions in the `Processor` class, comprising static methods for reading from, writing to the binary file, and adding new data entries. The `read_data()` method uses `pickle.load()` to load data from the file and handles different types of exceptions, providing a robust way to deal with file errors. If the file does not exist or is in an invalid format, the method returns an empty list.

The `write_data()` method utilizes `pickle.dump()` to save the investment data to the file. This action is performed each time a new entry is added, ensuring data persistence.

The `enter_new_data()` method solicits user input for the date and investment value, conducting error checking for an input format. Notably, the program sorts the data in descending date order each time a new entry is added, showcasing a design decision to maintain the data sorted rather than sorting it just before display.

The `main()` function is the entry point for the script. It employs a while loop to provide a continuously running menu to the user. It offers three options: reading and displaying the data, entering new data, and exiting the program. The reading operation sorts the data by date and displays the sorted data in the console. Each option is encapsulated within clear conditional statements, maintaining code readability and separation of concerns.

The script concludes with the `if __name__ == '__main__':` guard to ensure that the `main()` function is called when the script is run directly, but not if it is imported as a module. This aspect indicates a consideration for the potential future use of this code as a module in a more extensive program.

The Results in PyCharm

Figure 1 displays a test run of the data set as it appears with three total entries. The fairly simple operation shows the three resulting lines. While Figure 2 depicts the binary file results following such entry.

```
Menu:
1) Read data
2) Enter new data
3) Exit
Enter your selection: 1
******* The current Date/Investment Values are: *******
Date            Investment Value
2018-01-01      525.0
2016-01-01      125.0
****************************************

Menu:
1) Read data
2) Enter new data
3) Exit
Enter your selection: 2
Enter the date (yyyy-mm-dd): 2021-01-01
Enter the value of the investment: 555
Data saved successfully to AppData.dat
******* The current Date/Investment Values are: *******
Date            Investment Value
2021-01-01      555.0
2018-01-01      525.0
2016-01-01      125.0
****************************************

Menu:
1) Read data
2) Enter new data
3) Exit
Enter your selection:
```

Figure 1: PyCharm Operation

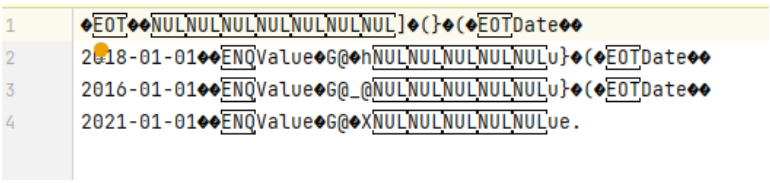

Figure 2: Binary File results following PyCharm entry

The Results in the Command Prompt

The command prompt operation (Figure 3) performed as expected and was identical to the PyCharm operation. The one additional line added is displayed in Figure 4.

```
C:\_PythonClass\Assignment07>python assignment07.py

Menu:
1) Read data
2) Enter new data
3) Exit
Enter your selection: 1
******* The current Date/Investment Values are: *******
Date                    Investment Value
2021-01-01              555.0
2018-01-01              525.0
2016-01-01              125.0
****************************************

Menu:
1) Read data
2) Enter new data
3) Exit
Enter your selection: 2
Enter the date (yyyy-mm-dd): 2022-01-01
Enter the value of the investment: 775.25
Data saved successfully to AppData.dat
******* The current Date/Investment Values are: *******
Date                    Investment Value
2022-01-01              775.25
2021-01-01              555.0
2018-01-01              525.0
2016-01-01              125.0
****************************************

Menu:
1) Read data
2) Enter new data
3) Exit
Enter your selection: 1
******* The current Date/Investment Values are: *******
Date                    Investment Value
2022-01-01              775.25
2021-01-01              555.0
2018-01-01              525.0
2016-01-01              125.0
****************************************
```

Figure 3: Command Prompt operation of the program

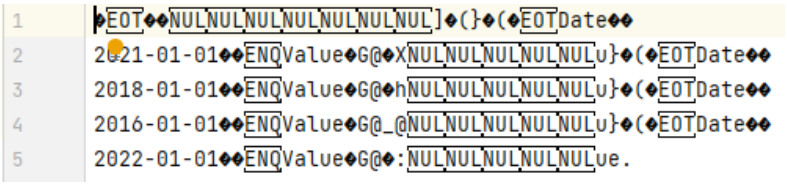

Figure 4: Binary file results follow command prompt operation

Summary

This Python script showcases software engineering principles: exception handling, separation of concerns, and module usage. Possible improvements include adding flexibility to the filename assignment, extending the menu functionality (e.g., removing or editing entries), and enhancing the user interface for a smoother user experience.

Overall working through the additional try-except operators is a great way to enhance the usability of one's code. This enhancement makes troubleshooting faster while completing the program performs more robustly.