Alison Langer

August 23, 2021

Foundations of Programming: Python

Assignment 07

Github Link: <https://github.com/alilan03/IntroToProg-Python-Mod07>

A Demonstration of Error Handling and Pickling

## **Introduction**

In this paper I will discuss the process of creating a to do list python script for assignment 07. The assignment requested a python script file that demonstrates the use of error handling and pickling for binary text files. This included four main concepts of user input/output, file I/O, try-except clauses, and pickling. This paper will begin with the initial file creation and will follow through to the final completion of the assignment with the functioning code.

### **Creating a Script File**

The first step for this assignment was to create a folder called “Assignment07” in the C: drive of the computer as a subfolder of the “\_PythonClass”. (Fig 1.1)

Graphical user interface, application

Description automatically generated

***Fig 1.1 Shows the “Assignment07” folder in the “\_PythonClass” folder***

The next step was to create a new PyCharm project in the “Assignment07” folder.

Graphical user interface, application

Description automatically generated

***Fig 1.2 Shows the “Assignment07.py” PyCharm file in the “Assignment07” folder***

The first step I took in writing the code for this assignment was creating a header to give a description of the project as well as provide a change log for updates to the code. (Fig 1.3)

Text

Description automatically generated

***Fig 1.3 The header for the “Assignment07” file***

### **Writing the Python Code**

To begin my project, I started by writing some lines of initial set up. This included importing the pickle module for binary file IO as well as declaring some variables in the data portion of my script. (Fig 2.1)

Text

Description automatically generated

***Fig 2.1 Shows the initial set up for the project***

Next I defined a method I called “process” which will process the user’s choice from the menu and perform the appropriate actions to fulfill their request. (Fig 2.2)

Text

Description automatically generated

***Fig 2.2 Shows the definition and docstring for the “process” method***

The “process” method consisted of a series of if-elif-else statements that performed the required actions for the user’s requests. The first if statement loaded the data, the second statement writes what the user specifies to the file, the third statement displays what is currently in the file and the else statement is a default clause for error handling. If the user tries to choose a menu option that is not available, the program will display a message saying, “That is not an option please try again…” and the program will revert back to the menu. For writing to the (Fig 2.3)

Text

Description automatically generated

***Fig 2.3 The if-elif-else statements that process the user’s menu choice***

For my second processing function, I defined a function called “read\_file” that will process the contents of the binary text file. By using the pickle import, I was able to call the “.load” function on pickle to read from the file. I also included a try-except clause here for error handling. From previous experience I know that programs frequently have problems when reading from an empty file. By using an “except EOFError” or a end-of-file error clause, I could print a separate message when the file was empty instead of the program crashing. (Fig 2.4)

Text

Description automatically generated

***Fig 2.4 The “read\_file” function that reads the contents of the binary text file***

Next, I wrote a two Input/Output functions for organization and to reduce redundancy. The first function was called “get\_choice” which essentially just prompts the user to choose a menu option and returns that choice to the function it was called from. (Fig 2.5)

Text

Description automatically generated

***Fig 2.5 The “get\_choice” function that prompts the user to choose from the menu***

The second Input/Output function was called “get\_text” which was responsible for prompting the user to enter some text to write to the binary file. The function then returned the user’s input to the function it was called from. (Fig 2.6)

Text

Description automatically generated

***Fig 2.6 The “get\_text” function that gets the user’s text to write to the binary file***

Finally, for my main portion of code I used a while loop to continually print the menu until the user chooses to exit the program. For this I separated choice 4 “Exit Program” in an if statement to check that condition before processing. But first, “get\_choice” was called to get the user’s menu choice. If the user didn’t want to exit, then the “process” function was called to perform the required actions to complete the user’s request. (Fig 2.7)

Text

Description automatically generated

***Fig 2.7 The main while loop that prints the menu with initial processing***

### **Running the Script**

The final portion of the assignment was to run the Python script in both PyCharm (Fig 3.1 - Fig 3.4) as well as a shell window (Fig 3.5 – Fig. 3.8) and record the running functionality.

Text

Description automatically generated

***Fig 3.1 Menu option 1: loading the data from the binary file***

Text

Description automatically generated

***Fig 3.2 Menu option 3: displaying the contents of the binary file***

Text

Description automatically generated

***Fig 3.3 Menu option 2: writing to the file***

Text

Description automatically generated

***Fig 3.4 Menu option 4: exiting the program***

Text

Description automatically generated

***Fig 3.5 Menu option 1: loading the data from the binary file***

Text

Description automatically generated

***Fig 3.6 Menu option 3: displaying the contents of the binary file***

Text

Description automatically generated

***Fig 3.7 Menu option 2: writing to the file***

Text

Description automatically generated

***Fig 3.8 Menu option 4: exiting the program***

The last instruction was to verify that the program did write the given information to the specified “BinaryText” text file. This was completed by opening the text file that was created by the program to verify its contents. The contents were there, although unreadable in binary format. (Fig 3.3)

Graphical user interface, text, application

Description automatically generated

***Fig 3.11 The contents of the “BinaryText” text file***

### **Summary**

In this paper discussed the process of creating the to do list script in Python for assignment 07. This script demonstrates the usage of both error handling and pickling for binary text files. This included four main concepts of user input/output, file I/O, try-except clauses, and pickling. Input is useful for obtaining data from the user to utilize in the program. Output is useful for displaying messages or prompts to the user. File I/O can be used to either read or write to a file. In this case we read the binary text from the file and wrote the user’s input to the file as well. Error handling was demonstrated through try-except clauses that were used to detect if the file was empty before reading. For my research on error handling, I used the website <https://docs.python.org/3/tutorial/errors.html> (External Site). Pickling was demonstrated through binary text file IO using the “pickle” import to both load and dump text from the file. For my research on pickling, I used the website <https://www.geeksforgeeks.org/understanding-python-pickling-example/> (External Site). Throughout my paper I discussed the steps and logic behind each of my decisions while coding the Python script for this assignment and concluded with the final display of the script running in a shell window.