Kristen Doehlert

August 24, 2022

ITFDN 110 B

Assignment 07

<https://github.com/kdoehlertUW/IntroToProg-Python-Mod07>

Pickling and Error Handling

# Introduction

In module 7 we learned how to use pickling to save data to and load data from a binary file. We also learned how to use structured error handling to manage errors and provide more user-friendly error messages. In this paper I am going to discuss how I created an Expense Tracking script that uses both pickling and error handling.

# Overview of the script

The Expense Tracking script has a similar structure to the To Do List script from module 6, but instead of managing lists of dictionaries with a Task and Priority, it manages lists of lists with an Expense Name and Amount. The script starts by loading data from a binary file, if one exists. The user is presented with a menu of options: 1. Add a new Expense 2. Remove an existing Expense 3. Save Data to File 4. Calculate total Expenses 5. Exit Program.

Option 4 uses two brand new functions for processing and IO to print the sum of all expenses. Instead of reading and writing lines of data into a plain text file, the list object is pickled and unpickled. Structured error handling is included in three functions in the script.

# Pickling data

When menu option 3 is entered, the function *Processor.pickle\_data\_to\_file* is called. This function opens a file in binary write mode using the file name that is passed in, pickles the list object that is passed in, and stores that pickled object in a binary file. Figure 1 shows the code used to create this function.

Text

Description automatically generated

***Figure 1. Function to pickle the data***

When you open the binary file in a text editor, you can see that the content is obscured. In Figure 2 you can see that the Expenses ‘Groceries’ and ‘Water’ are included, but you cannot easily read any other information.

A picture containing graphical user interface

Description automatically generated

***Figure 2. Pickled data opened in text editor***

# Error handling

Structured error handling was added in two functions where user input was collected using try-except blocks.

The first IO function with error handling is *input\_menu\_choice*. For this function I created a custom exception class called MenuSelectionRange. This class returns the message ‘Menu selection must be 1-5’ when printed. Figure 3 shows the code used to create this class.

Graphical user interface, text, application

Description automatically generated

***Figure 3. Class for custom exception used if menu selection is out of range***

The MenuSelectionRange error is raised if the user input is not 1, 2, 3, 4, or 5. When this error is raised, the error message is printed. The invalid selection is returned and no if statements are triggered, so the script returns to the beginning of the while loop where *IO.input\_menu\_choice* is called again, and another menu selection can be inputted. Figure 4 shows the code for this function.

Graphical user interface, text, application

Description automatically generated

***Figure 4. Function to get user input for menu selection***

Figure 5 shows how this error looks when the script is run in Command Prompt. An incorrect value (8) is entered, the error message is printed, and the script returns to the beginning of the loop where it prints the expenses and menu options.

Text

Description automatically generated

***Figure 5. Error handling when an incorrect menu selection [8] is made***

A try-catch block was also added to the IO function *input\_new\_expense\_and\_amount*. The input for amount is converted to a floating-point value, and anything other than a number is entered a ValueError is raised. The catch block prints a custom error message informing the user that the expense and amount was not added, and that the amount must be a number. The try-catch block is wrapped in a while loop that continues until a number is assigned to the *local\_amount* variable. Figure 6 shows the code for this function.

Graphical user interface, text

Description automatically generated

***Figure 6. Function that gets user input for expense name and amount***

Figure 7 shows what this error handling looks like in the Command Prompt. A string (Too high) is entered as the amount, which raised the ValueError when trying to convert it to float. The custom error message is printed, and the user has another opportunity to enter the Expense Name and Amount until a valid entry is made.

Text

Description automatically generated

***Figure 7. Error handling when an incorrect value [Too high] is entered for amount***

# Unpickling with error handling

The Processor function *load\_data\_from\_file* combines unpickling and structured error handling. The code to load the pickled list and assign it to *list\_of\_rows* is wrapped in a try-except block. This is necessary because the file is being opened in “rb” mode and will raise an error if the file does not exist. With the try-catch block added, an error message with be printed if the file does not exist, but the script will continue. Figure 8 shows the code using both unpickling and error handling.

Text

Description automatically generated

***Figure 8. Function to unpickle the data wrapped in a try-except block***

Figure 9 shows the error message that is printed when the script is run but the binary file does not exist.

Text

Description automatically generated with low confidence

***Figure 9. Error message that is generated if there is no binary file to load from***

# Summary

In this paper I discussed how I created a script that stores a list of data related to expenses by pickling it to the binary file. The script also used try-except blocks to handle common errors that could occur when running the application.