Erickson Smith
December 01, 2020
IT FDN 110 A Au 20: Foundations of Programming: Python
Assignment 07

Structured error handling and Pickling

Introduction

In this week's assignment. Assignment 07 we learned the importance of structured error handling. Error handling can improve your code and make life a lot easier as a programmer. Pickling is another subject we covered this week. With pickling you can save a complex piece of data and save it into a binary file. In this paper I will go over my steps and thoughts.

Starting the code

In order to start this week's code I needed to first look up those two topics for the assignment. One being structured error handling and the second being pickling.

Structured Error Handling is a way of making everyones lives a lot easier by catching errors before they cause your code to blow up. As a programmer errors are an everyday occurrence but you can use (try and except) blocks to smooth those errors out. Instead of being stopped in its tracks your code will continue on. Such as a number being put in a name or a menu with 3 options and the user inputs 7. This is the type of error handling I implemented this week. (Figure 01).

```
class NotValidChoice(Exception): # Exception that reminds the user to choose option 1-4
    def __str__(self):
        return "Please enter valid option: (1 - 4)"
```

Figure 01: Function to catch invalid user inputs

"Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization" (Figure 02).

```
@staticmethod
def pickle_to_file(lst_table, file_name):
    """Function for pickling a list of dictionary key / value
       pairs to a file."""
    file = open(file_name, "ab+")
    pickle.dump(lst_table, file)
    file.close()

@staticmethod
def unpickle_from_file(file_name):
    """Function for unpickling a list of dictionary key / value
       pairs from a file."""
    file = open(file_name, "rb")
    list_of_data = pickle.load(file)
    file.close()
    return list_of_data
```

Figure 02: Functions for Pickling and Unpickling

Once I had those two subjects figured out I was ready to start on the rest of my code.

Building the code

To start with the rest of the code I need to set my variables that I would be using. (Figure 03).

```
file_name = "Gamelist.txt"
menu_choice = None
video_game = ""  # element in game rating dictionary
game_rating = ""  # element in game rating dictionary
dict_row = {}  # game row (Title and Rating)
lst_table = []  # list of games and their ratings
```

Figure 03: Initializing variables

Next I needed to make my menu for the user to interact with. This menu has four options and will only work with those four options thanks to error handling. (figure 04).

```
@staticmethod
def print_menu_option():
    print("""
    Main Menu:
    1.) Add Favorite Games
    2.) Save Data
    3.) Load Data
    4.) Exit
    """)
```

Figure 04: Menu of Options.

Option 1, Add favorite games. This option prompts the user to enter a favored game of theirs and a rating that they would give to said game.This is all done with the help of a function called input_video_game.(figure 05).

```
@staticmethod
def input_video_game():
    """ Gets video game and rating from user """

    game = input("Enter Video Game: ")
    rating = input("Enter Video Game's rating (Great, Good, Bad): ")
    return game, rating
```

Figure 05: Function for collecting data from users.

As you can see this function collects data from the user by asking for a video game name and a rating that would bestow that game then it returns the data. In order for that data to go anywhere I created another function that would work with the previous function. This next one would format the data and put it into an easy to read dictionary as a row of data creating a table of data. (figure 06.)

```
@staticmethod
def add_game_rating(game, rating, list_of_rows):
    """ Adds video game and rating to dictionary as a row.
        Then adds dictionary to list (table).
    """

    row = {"Game": game.strip().capitalize(), "Rating": rating.strip().capitalize()}
    list_of_rows.append(row)
    return list_of_rows
```

Figure 06: Function for formatting data into a dictionary.

After that I needed to work on the main body of the script. Starting with a while loop that would allow the program to continuously run until the user was finished with it. The first user option allows the user to input their desired game and rating. (figure 07.)

```
try:
    if menu_choice == "1":  # Add Game/Rating to list
        video_game, game_rating = IO.input_video_game()
        Processor.add_game_rating(video_game, game_rating, lst_table)
        input("Press [Enter] to continue")
        continue
```

Figure 07: Menu option 1 adding data to the list.

For option two I needed to implement the pickling function I defined earlier. Then tell the user that their data was saved.(figure 08.)

```
elif menu_choice == "2":# Save list Data to Binary File
    Processor.pickle_to_file(lst_table, file_name)
    input("Data saved! Press [Enter] to continue")
    continue
```

Figure 08: Menu option 2 pickling to a binary file.

Option three loads the data that was just saved into a binary file back into memory that humans can read. (figure 09.)

```
elif menu_choice == "3":# Loads Data from Binary File
    Processor.unpickle_from_file(file_name)
    print(lst_table)
    continue
```

Figure 09: Menu Option 3 loading data from a binary file.

Option four allows the user to say they are done manipulating the data and have the program close. (figure 10.)

```
elif menu_choice == "4":# Exits the program
    break
```

Figure 10: Menu option 4 Exits the program.

Now I needed to take the program one step further and make sure the users did not mistakenly enter a number (or letter) that wasn't part of the menu options. This is where error handling comes into place. (figure 11.)

```
    else:
        raise NotValidChoice()

except ValueError as e:
    print("Please enter a number (1-4)")
    print(e, "\n")

except FileNotFoundError as e:
    print("File was not found.")
    print(e, "\n")

except Exception as e:
    print(e, "\n")
```

Figure 11: Error handling.

With the NotValidChoice Function I created earlier in the program anything other than 1-4 is seen as an error but it does not cause the program to stop running. It will return an error message to the user and pick up at the last step the program was on.

Summary

In this week's assignment I learned how powerful the process of error handling can be. The ability to foresee complications and how to account for them can make things run so much more smoothly for not only the programmer but the user as well. Pickling can help in keeping files somewhat secure in that it stores data in an obscured fashion.