Kristin Landgren Martinez
December 9, 2021
Foundations of Programming: Python
Assignment 07
GitHub Repo
GitHub Website

Pickling & Structured Error Handling

Introduction

This assignment and module introduced custom error handling, more advanced file handling, and the use of pickling and binary files. The goal of the assignment is to illustrate pickling and structured error handling, and to practice independent research. The results are posted to a new GitHub website in addition to the standard documents.

Concepts

Binary Files & Pickling

Binary files contain data stored as bytes, not as text characters, and require specific applications to be read. Because they encode custom data types, binary files are not readable by eye like text files are. However, in Python this also means that they can be used to store an entire data object instead of reading it in and out if its native format to a text file. Picking is the name of the Python process used to preserve, or serialize, a complex data object so that it can be unpickled and used later in its original form.

File Handling

For both text and binary files, there are built-in Python commands for basic file handling tasks such as opening, reading, writing, appending, and closing. While these can be used ad hoc, building them into custom functions allows you to address various common scenarios and easily re-use the code elsewhere. Best practices such as closing the file after use are easy to build into a function. In addition, creating custom functions and a using a separation of concerns structure facilitates clear error handling.

Structured Error Handling

Structured error handling is a way of organizing code that might introduce errors into a section to try, followed by one or more "except" sections to move through if the try block gives an error. Each except section can be set up to capture an error type, from specific to more general, and execute code customized to that error type. This format is much more helpful and polished than the alternative, in which the program crashes and the user receives Python's auto-generated error messages. While these can still be captured and shown to the user in structured error handling, using try/except blocks allows the errors to be understood in the context of the specific program and task. Python error information is contained in the Exception class, and a developer can add a custom exception to this class which will behave like a Python error. This is typically done to handle cases they want to disallow in their script, but which would not raise a Python-generated error.

RSVP Script

This script helps the user track the RSVP list for an upcoming party. The user has three options, which are repeated until the exit option is selected: they can enter a guest name and number, view the current RSVP list, or save and exit. The RSVP entries are stored as a list of dictionary rows, and at the end the entire list is pickled to a binary file called RSVP.dat. If the file already exists in the directory when the user begins the program, it unpickles the list and starts to add to it. The script incorporates structured error handling using built-in and custom exceptions to make it easier for the user to understand and correct course when something goes wrong.

Setup

I planned out the program using pseudocode and the general menu selection structure of Assignment06. I had a general plan for where I would use structured error handling and pickling. However, through the process of writing and code and running into errors myself, the final product took a much different track.

Figure 1: Pseudocode with initial functions, test run

Development

As I built out the script, I re-used functions from similar scripts such as Assignment06 and the Module07 labs whenever possible. However, even with relatively similar content a number of adjustments were needed to make the script run as expected. The use of pickling made some steps much easier, such as loading data from the file. But other elements of pickling only became clear to me as I ran up against errors. Corrections and improvements I made included:

- Opening the pickled file in write mode rather than append mode, because having a single list of dictionary rows was best for the step where I summed the guests. I hadn't realized at first that appending added another list object rather than adding to the existing list. In the

final version, the single list is unpickled (if present) and loaded to a list object, appended to if the user adds RSVPs, and then saved back to the file in its entirety, overwriting the original.

- Adding an if/else block in the Save & Exit step to catch cases where the list of rows to be saved is still empty, to avoid saving an empty file. This would occur if there was no pre-existing file and the user had not entered data before selecting Save & Exit. I tried several ways to do this, including a try/except block, but this seemed like a better way be able to always initialize the list variable but only save if there was data in it.

- Adding a third menu option to view the details of the list, not just the total guests. This seemed like a useful feature, and good practice presenting the list of dictionary rows. I elected not to add a case for no data, since printing the sum of 0 seed sufficiently clear.
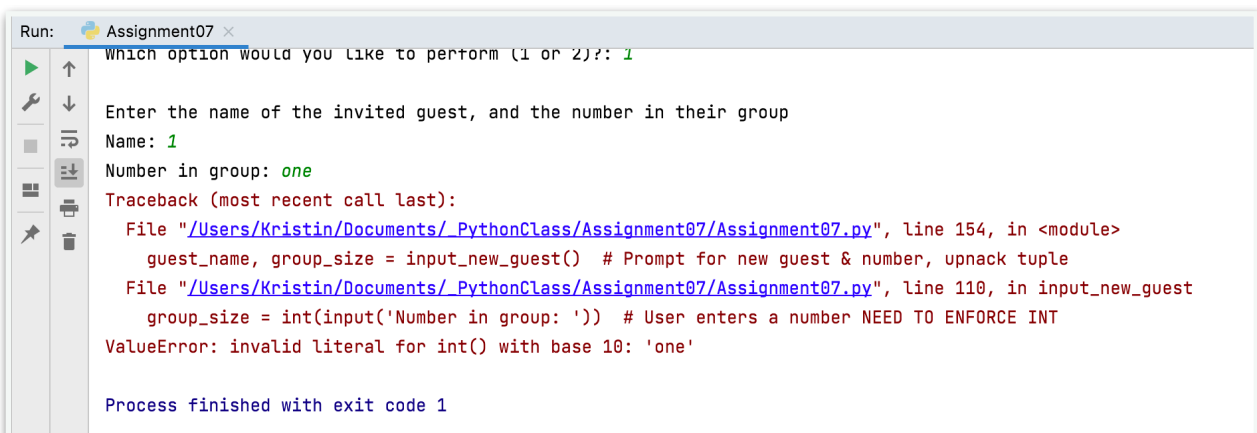
Error Handling

I added most of the error handling towards the end of my work on the script, primarily based on errors I received. The final version of the script uses try/except blocks to catch exceptions and present custom messages in the following cases:

FileNotFoundError: used at the beginning of the script to differentiate runs where the file RSVP.dat is already present from runs where it is not. In the latter case, the user gets a message that this will be a new list. The file will be created at the end of the program if the user has entered data. The code for this block in the main body of the script is below.

```python
# Try/Except block to check if the file already exists in the same directory
try:  # If the file exists, unpickle it and load the list to the object
      rsvp_lst
    rsvp_lst = unpickle_data_from_file(rsvp_file)  # Unpickle the list do that
        it can be added to
    output_existing_list_confirmation()  # Confirm to the user that the
        existing list will be added to
    output_total_guests(sum_guests(rsvp_lst))  # Print a reminder of how many
        guests are on the list
except FileNotFoundError:  # If the file does not yet exist, print a message
        and move on
    output_new_list_confirmation()  # Confirm to the user that this will be a
        new list
```

ValueError: For the sum function to work, the user must enter group_size as an integer. I used int() when defining group_size, but a text entry like "Mary" throws a ValueError. This exception returns the user to the menu to try again.

Figure 2: ValueError text from Python

```
Run:    Assignment07 ×
        Which option would you like to perform (1 or 2)?: 1

        Enter the name of the invited guest, and the number in their group
        Name: 1
        Number in group: one
        Traceback (most recent call last):
          File "/Users/Kristin/Documents/_PythonClass/Assignment07/Assignment07.py", line 154, in <module>
            guest_name, group_size = input_new_guest()  # Prompt for new guest & number, unpack tuple
          File "/Users/Kristin/Documents/_PythonClass/Assignment07/Assignment07.py", line 110, in input_new_guest
            group_size = int(input('Number in group: '))  # User enters a number NEED TO ENFORCE INT
        ValueError: invalid literal for int() with base 10: 'one'

        Process finished with exit code 1
```
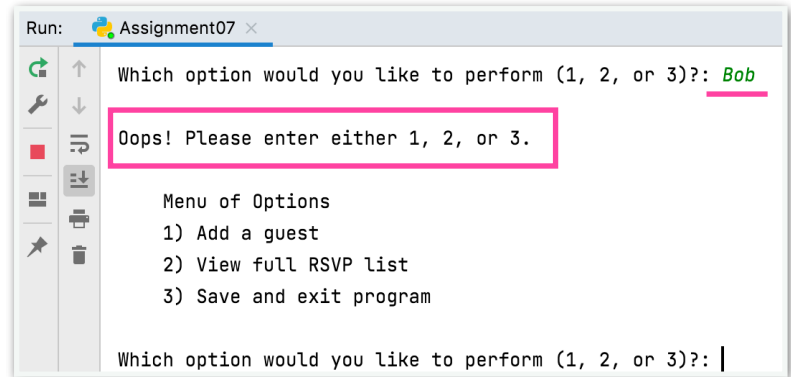
CustomException_UserChoice: I created a custom exception to display a reminder message when the user selection is not in the menu range.

Figure 3: Custom Exception in code and PyCharm

```python
class
CustomException_UserChoice(Exception):
    """ Custom error message to raise
if the user choice is not 1, 2, or 3

    :return: nothing
    """

    def __str__(self):
        return 'Oops! Please enter
            either 1, 2, or 3.'
```

```
Run:    Assignment07 ×

    Which option would you like to perform (1, 2, or 3)?: Bob

    Oops! Please enter either 1, 2, or 3.

        Menu of Options
        1) Add a guest
        2) View full RSVP list
        3) Save and exit program

    Which option would you like to perform (1, 2, or 3)?: |
```

Running the Script

After a great deal of troubleshooting, the script ran as expected in both PyCharm and terminal. A list can be started and saved in one program, and added to in the other, as shown in the progression below. The script handles several likely error cases, and also adjusts if the file is empty.
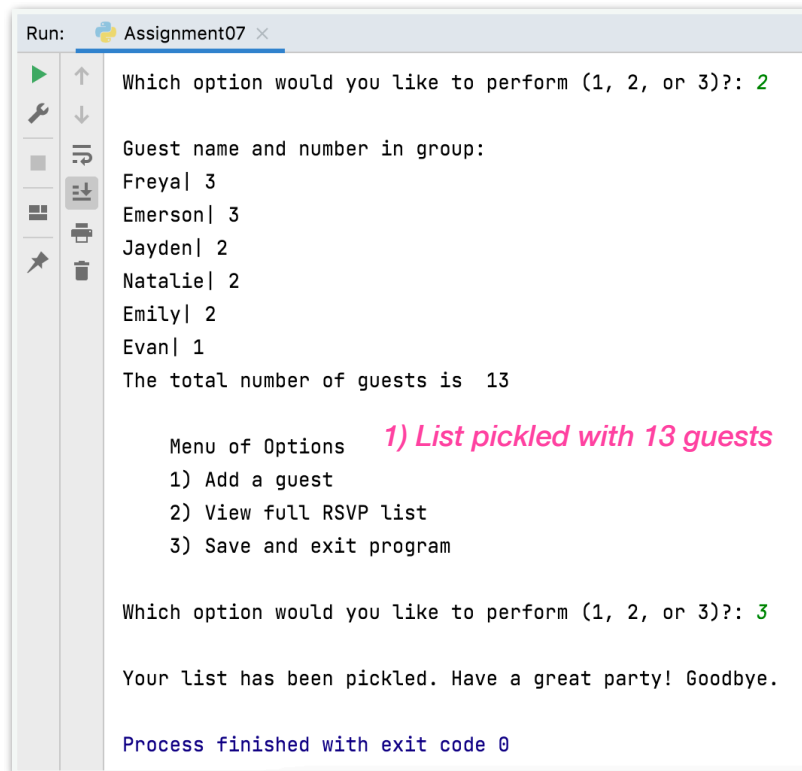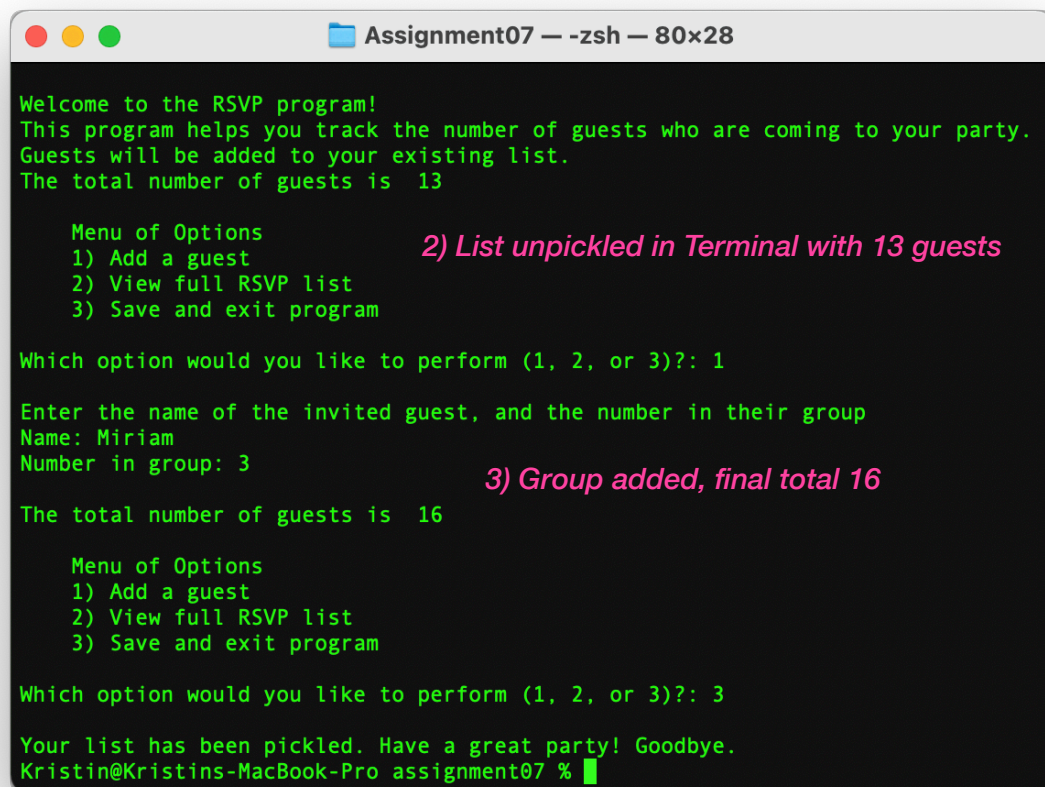
Figure 4.1: RSVP file picked from PyCharm

```
Run:    Assignment07 ×

    Which option would you like to perform (1, 2, or 3)?: 2

    Guest name and number in group:
    Freya| 3
    Emerson| 3
    Jayden| 2
    Natalie| 2
    Emily| 2
    Evan| 1
    The total number of guests is  13

        Menu of Options      1) List pickled with 13 guests
        1) Add a guest
        2) View full RSVP list
        3) Save and exit program

    Which option would you like to perform (1, 2, or 3)?: 3

    Your list has been pickled. Have a great party! Goodbye.

    Process finished with exit code 0
```

Figure 4.2: RSVP file unpickled in Terminal, guest added


```
Welcome to the RSVP program!
This program helps you track the number of guests who are coming to your party.
Guests will be added to your existing list.
The total number of guests is  13

    Menu of Options
    1) Add a guest               2) List unpickled in Terminal with 13 guests
    2) View full RSVP list
    3) Save and exit program

Which option would you like to perform (1, 2, or 3)?: 1

Enter the name of the invited guest, and the number in their group
Name: Miriam
Number in group: 3               3) Group added, final total 16

The total number of guests is   16

    Menu of Options
    1) Add a guest
    2) View full RSVP list
    3) Save and exit program

Which option would you like to perform (1, 2, or 3)?: 3

Your list has been pickled. Have a great party! Goodbye.
Kristin@Kristins-MacBook-Pro assignment07 %
```

Conclusion

Developing this script to illustrate pickling and structured error handling taught me about these concepts and others, and gave me important practice debugging and thinking through my code. In addition, I learned to present documents like this one on GitHub pages using Markdown. The tasks for this module have helped me feel more confident planning, developing, and presenting code on my own.

Appendix: Useful links

Pickling

https://www.datacamp.com/community/tutorials/pickle-python-tutorial

https://realpython.com/python-pickle-module/

https://www.geeksforgeeks.org/how-to-update-a-pickle-file-in-python/

https://sites.pitt.edu/~naraehan/python3/pickling.html

https://betterprogramming.pub/dont-fear-the-pickle-using-pickle-dump-and-pickle-load-5212f23dbbce

Structured Error Handling

https://docs.python.org/3/tutorial/errors.html

https://docs.python.org/3/library/exceptions.html#bltin-exceptions

https://realpython.com/python-exceptions/

https://betterprogramming.pub/handling-errors-in-python-9f1b32952423

https://www.datacamp.com/community/tutorials/exception-handling-python