

James Henderson

8/24/2022

IT FDN 110 B

Assignment07

<https://github.com/hendej25/IntroToProg-Python-Mod07>

<https://hendej25.github.io/IntroToProg-Python-Mod07/>

Assignment 7 – Menu Script

Introduction

For this assignment, I worked to modify an existing script similar to the one that we completed for processing a list of tasks and their priorities. I used the base of that code to represent an interactive restaurant menu that can be edited and saved to a file in a binary format using the pickle module. The script is separated into 3 distinct layers – processing, I/O, and a main code section. The script also has several areas that illustrate the use of error handling in Python.

Menu Script: Processing Binary Data using Pickle

To accomplish the goals of this assignment, I created a class called PickleJuice which performs the processing tasks. The functions that read / write data out to a flat *.dat file use the pickle module, since that data uses binary formatting; the other functions in this class modify data in the program's memory.

```
class PickleJuice:
    """ Performs processing tasks using the pickle module """

    @staticmethod
    def create_new_file(file_name):...

    @staticmethod
    def read_data_from_file(file_name):...

    @staticmethod
    def save_data_to_file(file_name, menu):...

    @staticmethod
    def add_item_to_menu(name, price, category, description, menu):...

    @staticmethod
    def remove_data_from_menu(name, menu):...
```

Figure 1 – The processing layer of the script is organized into the PickleJuice class

In **Figure 2**, you can see how the pickle module is used to read in an entire list of Dictionary objects at once. Even though the `load` function only loads a single line at a time from the text file, and the menu list may contain multiple objects, the entire list is treated as if it were a single line – so there is no need to iterate through during the load process. Saving the data is equally simple, using the `dump` function.

```
@staticmethod
def read_data_from_file(file_name):
    """ Reads binary data from a file that was stored with pickle.

    :param file_name: (string) with name of file
    :return:
        menu - (list) of data rows
        status - (string) indicating function status
    """
    menu = [] # initialize menu list
    with open(file_name, "rb") as file:
        menu = (pickle.load(file))
        status = SUCCESS_STR

    return menu, status

@staticmethod
def save_data_to_file(file_name, menu):
    """ Saves a binary representation of Python object data into a file using pickle.

    :param file_name: (string) with name of file
    :param menu: (list) containing data to be written out:
    :return:
        menu - (list) of data rows
        status - (string) indicating function status
    """
    with open(file_name, 'wb') as file:
        pickle.dump(menu, file)
        status = SUCCESS_STR

    return menu, status
```

Figure 2 – Using the `pickle.load()` and `pickle.dump()` functions to read and write binary data

Menu Script: Input / Output

As I mentioned in the introduction section, the script for this assignment has been separated into 3 distinct layers. The Input / Output functions that display data to the user, or take in input from the user, are organized into a Class named "IO". There are eight functions in all – of these, 2 just display data or choices to the user, while 6 are interactive in that they require some type of input from the user.

The interactive functions (names beginning with “input_”) return the user input via one or more variables. Within the main code body, these returned values are then handed off to one of the Processor-class functions as needed.

```
class IO:
    """ Performs tasks related to the interactive portion of the program"""

    @staticmethod
    def print_user_choice_menu():...

    @staticmethod
    def create_new_file(file_name):...

    @staticmethod
    def input_press_to_continue(optional_message=''):...

    @staticmethod
    def input_user_menu_choice():...

    @staticmethod
    def print_restaurant_menu(menu, sort_by_key1='', sort_by_key2=''):...

    @staticmethod
    def input_yes_no_choice(message):...

    @staticmethod
    def input_new_menu_item():...

    @staticmethod
    def input_menu_item_to_remove():...
```

Figure 3 – There are 8 different static methods for handling interaction with the user

One of the areas in the script where error handling is used is inside the print_restaurant_menu function (**Figure 4**). This function has the ability to sort the menu (which is a list object) by different characteristics of the items on the menu (which are Dictionary keys). However, if the function is passed sort keys that don't actually correspond to the names of the Dictionary keys in the list, an exception will be raised – the user is notified of the error and no sort is applied.

```

@staticmethod
def print_restaurant_menu(menu, sort_by_key1='', sort_by_key2=''):
    """ Shows the current restaurant menu.

    :param menu: (list) of restaurant menu items
    :param sort_by_key1: (string) optional key to sort by: Name, Price, Category, etc.
    :param sort_by_key2: (string) optional 2nd key to sort by: Name, Price, Category, etc.
    :return:
        None
    """
    # sort the menu items for display (or not) according to the sort key arguments
    try:
        if sort_by_key1 and not sort_by_key2:
            sorted_menu = sorted(menu, key=lambda d: d[sort_by_key1])
        elif sort_by_key1 and sort_by_key2:
            sorted_menu = sorted(menu, key=lambda elem: "%s %s" % (elem[sort_by_key1], elem[sort_by_key2]))
        else:
            sorted_menu = menu
    except Exception as e:
        print("The program tried to sort the menu items on a characteristic that doesn't exist. "
              "A default sort order was applied.")
        sorted_menu = menu

```

Figure 4: Raising an error if invalid sort keys are passed to the print_restaurant_menu function

Menu Script: Custom Error Classes

The script also contains a few custom error classes for errors that are explicitly raised in the main code body section (**Figure 5**):

```

class UserCancelledNewFileCreation(Exception):
    pass

class InvalidUserMenuChoice(Exception):
    pass

class InvalidSaveChoice(Exception):
    pass

```

Figure 5 – Custom error classes for better identification of custom errors

As you can see in **Figure 6**, these classes can be visually helpful for other individuals editing the code to see what the specific error being raised is related to:

```

        except InvalidSaveChoice as e:
            print("Please enter a valid choice ('y' to save, 'n' to cancel).")

        continue # to show the user menu

    elif choice_str == '5': # Exit Program
        print("Goodbye!")
        break # and exit

    else: # prompt the user for a valid menu choice if an invalid choice was entered
        raise InvalidUserMenuChoice("Invalid menu choice.")

except InvalidUserMenuChoice: # if invalid choice, let the user know
    print("Please enter a valid choice [1-5].")

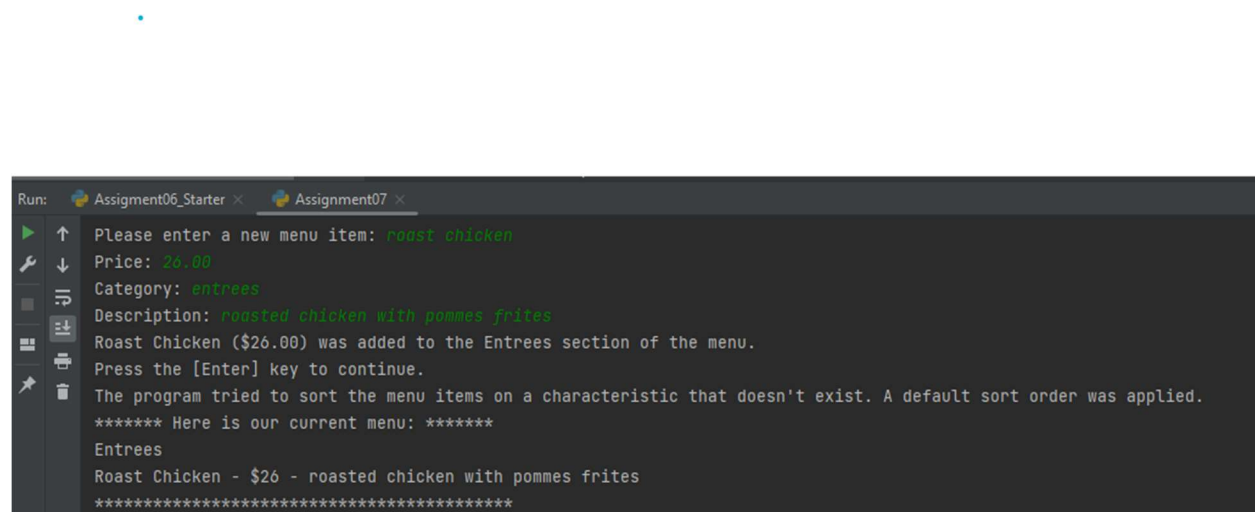
```

Figure 6 – Using the custom error classes to raise errors in the main code body

Menu Script: Running the Script

To test the script, I first ran the Assignment07.py file within PyCharm, using the latest available Python interpreter on my machine (Python 3.10).

You can see part of the script's input / output in process in **Figure 7** below – I intentionally added a bug to the main code body to ask the menu to print on a key that wasn't one of the Dictionary keys being used for the menu items, to show how the error handling would work. The user is notified of the issue, and the menu is printed with a default sort order (no sort) instead:



```

Run: Assignment06_Starter x Assignment07 x
Please enter a new menu item: roast chicken
Price: 26.00
Category: entrees
Description: roasted chicken with pommes frites
Roast Chicken ($26.00) was added to the Entrees section of the menu.
Press the [Enter] key to continue.
The program tried to sort the menu items on a characteristic that doesn't exist. A default sort order was applied.
***** Here is our current menu: *****
Entrees
Roast Chicken - $26 - roasted chicken with pommes frites
*****

```

Figure 7 – Testing the error handling when a bug was intentionally added to the code

I also ran the script from the Windows Command Prompt window (**Figure 8**). I verified that data from a previous run of the script was correctly picked up at the start of the program.

```
C:\_ PythonClass\Module7>python.exe assignment07.py
'C:\_PythonClass\Module7' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\hende>cd C:\_PythonClass\Module7

C:\_PythonClass\Module7>cd assignment07

C:\_PythonClass\Module7\Assignment07>python.exe assignment07.py
Data loaded successfully.
Press the [Enter] key to continue.
***** Here is our current menu: *****
Beverages
*****
Iced Tea - $3 - classic iced tea

Entrees
*****
Roast Chicken - $26 - roast chicken with pommes de terres
Steak Frites - $24 - steak frites with hollandaise sauce

Menu of Options
1) Add a new menu item
2) Remove an existing menu item
3) Save menu to file
4) Reload menu from file
5) Exit program

Which option would you like to perform? [1 to 5] -
```

Figure 8 – running the script from the Windows command prompt

Summary

In this assignment, I edited an existing script to make it work with 3 layers – a processing layer (one class), interactive layer (one class), and a main code body. I made sure that the functions in the script were all appropriately documented with docstrings, and that the code I added was compartmentalized correctly. I also demonstrated the use of pickle to read/write data in a binary format, as well as basic error handling throughout the script.