Brian Christopherson

28-MAY-2025

Python 100

Assignment06

<https://github.com/cb658/Python110-Spr2025>

A program using classes and functions, with Separation of Concerns patterning

# Introduction

This paper will discuss the process used to create Assignment06, which involves writing a Python program that registers new students and adds them to the student enrollment. To do this, the program reads the enrollment file and writes the user-input data back to the enrollments file in the JavaScript Object Notation (JSON) format.

The program presents a menu to the user and uses a *while* loop and conditional logic to control the flow of the program. The program uses constants, variables, dictionaries and a list to store and display the data. Classes and functions are used to organize the program, and the program is further organized in patterns based on the Separation of Concerns design principle.

The program also uses the try-except structured error handling method.

This program was written in the PyCharm IDE. Documentation of the program’s testing can be found in Appendix A; the full code is in Appendix B.

# Writing the Python program

The program creation process began with a review of the acceptance criteria for the program. The acceptance criteria included using a specific name for the program file, a script header and the use of specific constants and variables. The constants were strings presenting a multi-line menu and the name of the file to be used for the read and write operations. For the multi-line menu, a triple-quoted string was used. The menu presented four choices to the user.

The acceptance criteria defined input and output options based on the user choices from the menu:

* The first choice prompts the user via an input() function to enter the student’s first and last names, and the name of the course for registration. This information was to be written to the students list of lists.
* The second menu choice uses the print() function to display the collected data in the students variable back to the user in a comma-separated string.
* The third menu choice opens a file in write mode using the open() function, writing the data to a JSON file, then closing the file to allow future input. The program also uses the students variable to display what was written to the file.
* The fourth menu choice exits the program.

The code contained a while statement to loop through the various input choices. The looping allows for another of the acceptance criteria, which was to allow the user to enter multiple student registrations, if desired, and store all those registrations.

The acceptance criteria defined some structured error handling at various points within the program execution: when the file is read in, the user’s input of the first name, the user’s input of the last name, and lastly when the dictionary rows are written to the file. The last requirements of the acceptance criteria required specific tests of the program input and output functions, as well as confirming the program runs correctly in both PyCharm and the console.

# Creating the script file

Creation of the program began by opening the assignment starter file in PyCharm. That file was then saved as Assignment06.py and the change log section of the script header was updated. Figure 1 shows the script header used.

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 1 - Script Header**

# The Program Constants and Variables

The assignment starter file contained simple pseudo-code to guide the creation of the code body. This pseudo-code was leveraged to define the constants and variables, the input and output sections, and the flow of the program. Type hints were used to identify the data types used in the constants and variables.

The constants in the program were the previously mentioned user menu (a string encapsulated in triple quotation marks), and the file name for processing the data. This was also a string with the value of “Enrollments.json”. Both consonant names were written in uppercase, to comply with Python coding standards. The constants are shown in Figure ? below.

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 2- The program's Constants**

The data types for the variables used in the program were strings, an object for the file, a dictionary for the student\_data variable, and a list for storing the accumulated data on the students. The variables include variables for capturing the student name data and course name, as discussed previously. The acceptance criteria also defined a string named **menu\_choice** for storing the menu choices input by the user. All the variables were initially defined as empty; this involved using double quotation marks (“”) with no space between them for the strings, “None” object for the object data type, empty braces (**{}**) for the dictionary and empty brackets (**[]**) for the lists. Figure ? below shows the variable definitions.

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 3 - the program's Variables**

# Reading the json data file, with error handling enabled

As defined in the **FILE\_NAME** variable, a file named Enrollments.json, is read in at the start of the program. Prior to this, the json module is imported using the statement *import json*, and a *try:* block is begun to trap possible errors. As suggested in the acceptance criteria, a list in json format was pre-populated into the file. Within the *try:* block, the code attempts to open the file using the *open()* function with the “r” parameter, accessing the information in read-only mode. If the file is successfully opened, the *json.load() function* is used to read the file and convert it to the students list, then the file is closed using *file.close().*

The error handling in this section of code begins with an exception block for the *FileNotFoundError* exception. If the file is not found, this error is caught and a print statement is called, advising the user that the file needs to exist before running the script. Additionally, built-in Python error information is displayed, showing the error, the error’s documentation string (using the *\_\_doc\_\_* attribute) and type.

If the *FileNotFoundError* exception does not occur, the next block catches the general exception. If an exception other than *FileNotFoundError* occurs, a print statement indicates a non-specific error occurred and displays the error, the error’s documentation string (using the *\_\_doc\_\_* attribute) and type.

At the end of this *Try-Except* block is the *finally:* block. The *if* statement in this block tests whether the file is not closed and if so, closes the file. Figure ? below shows the import of the json module, *the Try-Except-Finally* block and the reading of the json file within the Try block.

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 4 - Initial Try Block with load of JSON file**

# Looping through the user input for processing

The program menu allows for the program to potentially be run an infinite number of times, unless the user chooses the option to exit the program. Thus, a *while* loop was used to control the program flow, using “*while True”*, and conditional *if, elif* and *else* statements.

At the beginning of the *while* loop, the *print()* function displays the **MENU** constant, which asks the user to select a choice from the menu. The input from the user is assigned to the **menu\_choice** variable. These input choices were evaluated as strings using if statements and the equals condition, written as “==”.

## If input choice “1” (Register a Student for a Course)

If the user’s choice is “1”, another Try-Except block is begun. The input() function is used to prompt the user to enter the student’s first name, last name and the course name; these values are assigned to the corresponding variables. The student\_first\_name and student\_last\_name variables are both checked to make sure that no numeric values are entered for either name. This is done using the if statements (“***if not student\_first\_name.isalpha():***” and (“***if not student\_last\_name.isalpha():***”). If a numeric value is entered, the *ValueError* exception is raised and advises the user that the name should not contain numbers. If the information is entered in alpha format, the input strings are then added to the **student\_data** dictionary in json format. Next, the **student\_data** dictionary is then added to the **students** list using the *append()* method. A *print()* statement confirms the student registration to the user. If the *ValueError* is raised during the student name input, the technical error message and document string are presented to the user. If any other error occurs, a general exception is raised, along with the technical error message and document string. Finally, the *continue* keyword is used to continue to the next iteration. Figure ? below shows the beginning of the *while* loop, the presentation of the menu to the user, and the *Try-Except block* within the first iteration of the while loop.

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 5 - the beginning of the loop, student\_data input prompts and Try-Except block**

## Else if choice “2” (Show current data)

If the user’s choice is “2”, represented as ***elif menu\_choice ==”2”:***, the *print()* function displays each registration to the user, with a “for loop” for each student dictionary item in the **students** list. The *continue* keyword is used to continue to the next iteration. This code is shown in Figure ? below.

A computer code with text

AI-generated content may be incorrect.

**Figure 6 - presenting the current data**

## Else if choice “3” (Save data to a file)

If the user’s choice is “3”, represented as ***elif menu\_choice ==”3”:***, a new *Try-Except* block is begun. Within this loop the ***Enrollments.json*** file is opened in write mode by calling the **FILE\_NAME** constant. The contents of the **students** listis written to the ***Enrollments.json*** file using the json.dump function, with “indent of 2” added to enhance the readability of the resulting output in the file. The file is then closed using *file.close().* Next, the *print()* function is used for each row in the **students** list table to indicate to the user what students are now enrolled in the respective courses. For code readability an escape character is added to the print() line.

To allow for the potential of issues with incorrectly formatted data, an *except* block is begun, capturing the TypeError exception and then prompting the user to check that the data is in a valid JSON format. Another *except* block will capture any other errors. Both *except* blocks display the technical error messages to the user, with additional Python error information provided for any general errors.

At the end of the *Try-Except* block within this *elif* condition is the *finally:* block. The *if* statement in this block tests whether the file is not closed and if so, closes the file. The *continue* keyword is used to move to the next iteration. Figure ? below shows the code used for saving the data to the file.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure - Saving the data to file

## Else if choice “4” (Exit the program), or other choices were entered.

If the user’s choice was “4”, represented as ***elif menu\_choice ==”4”:***, the loop was ended using the **break** keyword.

If the user entered any other character than the choices presented in the menu, a *print()* statement was called to instruct them to only choose one of the options “**1, 2, 3, or 4”**.

Once the loop has fully ended, a final *print()* statement is presented to the user, indicating the program has ended.

This code for the loop end and program exit is shown in Figure ?.

A screenshot of a computer code

AI-generated content may be incorrect.

**Figure 8 - Ending the loop and exiting the program**

# Testing the program

To test the program within PyCharm, the Run Module (F5) command was invoked. The program presented the menu, accepted the user input, then presented it and the read-in data when the ‘current data’ option was chosen. The program also saved the data to the Enrollments.json file and exited if the ‘exit program’ choice was called, thus meeting the acceptance criteria. The student error handling blocks were tested by changing the name of the Enrollments.json file and entering numerals when entering the student’s first and last names. The program was then tested by opening the Windows console and calling the Assignment05.py file. The program also ran successfully via the console. Screenshots showing the PyCharm and console tests can be found in Appendix A.

## Test observation to be addressed in future revisions

While testing the input of student names, it was discovered that entering an apostrophe in the name, sometimes used in Irish surnames such as O’Hara, would invoke the *ValueError* based on the .*isalpha()* method (see Figure ? below). This should be addressed in future revisions of the program.

A white screen with black text

AI-generated content may be incorrect.

Figure - Error due to apostrophe conflict with.isalpha string method

# Summary

This assignment covered several concepts, such as declaring constants and variables; using the *input()* and *print()* functions; data processing using dictionaries, lists and files in the JSON format. The program also introduced structured error handling. The program successfully accepted input from the program’s user, of a student’s first name and last name and course for enrollment. That input was then output to the user, then added to the existing “**students**” enrollments list, which was read and assigned memory when the enrollments file was opened. The updated “**students**” enrollments list was then successfully written to the enrollments file. Errors with the file, and errors with the input of the student’s first name and last name were addressed within the program. The program was successfully tested in the PyCharm IDE and the Windows Command Prompt console.

The testing of the program is shown in Appendix A and the full code is shown in Appendix B.

# Appendix A – Testing the program

## PyCharm testing

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 10 - Choice 1, Input of student’s first & last name & course**

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 11 - Choice 2. Display of student’s first & last name & course input**

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 12 - Choice 3**

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 13 - Undefined choice was made, then Option 4 chosen**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure - Testing the FileNotFoundError

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Testing student first name error- handling

A screenshot of a computer program

AI-generated content may be incorrect.

Figure - Testing student last name error-handling

## Console testing

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 17 - Console testing, choices 1 & 2**

A computer screen shot of a program

AI-generated content may be incorrect.

**Figure 18 - Console testing, choice 3, \*other, and choice 4**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure - Testing multiple registrations

A screenshot of a computer program

AI-generated content may be incorrect.

Figure - Displaying the multiple registrations from previous test and writing to file

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 21 - Review of Enrollments.json after all tests completed**

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Files uploaded to GitHub

# Appendix B – Full code