Brian Christopherson

20-MAY-2025

Python 100

Assignment05

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

A program reading and writing json file data, with structured error handling

# Introduction

This paper will discuss the process used to create Assignment05, 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.

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 Assignment05.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 2 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 3 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 4 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**

A loop was begun to transform the data. The first line of the loop took each row and removed any trailing or whitespace characters using the *strip()* method. This same line also called the *split()* method to split the string into a list, using a comma as the separator, denoted as ***“,”***. This data was passed to the **student\_data** dictionary. The next line of the loop indexed the **student\_data** variable data into three distinct list items. In Python indexing begins with value [0], so the items were set to student\_data**[**0**]**, student\_data**[**1**]**, and student\_data**[**2**]**. The last line of the *for* loop appended the **student\_data** list to the **students** list. The **students** list was a “list of lists” – in this case all the student data lists represented in one single list. The file was then closed using the *close()* method, to allow future operations on the file. Figure 4 below shows the applicable code described in this section.

**Figure 5 - Reading the source 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 was “1”, the input() function was used to collect the student’s first name, last name and the course name. The information was then added to the **student\_data** list; the **student\_data** list was then added to the **students** using the *append()* method. A *print()* statement confirms the student registration to the user. The *continue* keyword was used to continue to the next iteration. Figure 5 below shows the beginning of the while loop, the presentation of the menu to the user, and the first iteration of the while loop.

A computer screen shot of a program

AI-generated content may be incorrect.

**Figure 6 - the beginning of the loop**

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

If the user’s choice was “2”, represented as ***elif menu\_choice ==”2”:***, the *print()* function displayed each registration to the user, with a “for loop” for each row of data in the **students** list in string format. The *continue* keyword was used to continue to the next iteration. This code is shown in Figure 6 below.

A computer code on a black background

AI-generated content may be incorrect.

**Figure 7 - presenting the current data**

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

If the user’s choice was “3”, represented as ***elif menu\_choice ==”3”:***, the Enrollments.csv file was opened in write mode. Each row in the students list was added to the **csv\_data** string variable in string format, with the new line escape character added to the end of the row, to allow for additional entries. The **csv\_data** string variable was then written to the file, then the file was closed using the *.close()* method. The information that was written to the file is presented to the user, via the *print()* function that loops through each user in the **students** list and captures the index position data for the student’s first name, last name and the course name. The *continue* keyword was used to continue to the next iteration. This code is shown in Figure 7.

## 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 then 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 7.

A computer screen shot of text

AI-generated content may be incorrect.

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

# Testing the program

At the conclusion of writing the program in PyCharm, the Run Module (F5) command was invoked to test the program. 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.csv file and exited if the ‘exit program’ choice was called, thus meeting the acceptance criteria. The program was then tested by opening the Windows console and calling the Assignment04.py file. The program also ran successfully via the console. Screenshots showing the PyCharm and console tests can be found in Appendix A.

# Summary

This assignment covered several concepts, such as declaring constants and variables, using the *input()* and *print()* functions, and data processing using lists and files. 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 in comma-separated format, 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. 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 9 - Choice 1**

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 10 - Choice 2**

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 11 - Choice 3**

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 12 - Undefined choice was made, then Option 4 chosen**

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 13 - Review of Enrollments.csv in text editor**

## Console testing

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 14 - Console testing, choices 1 & 2**

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 15 - Figure 13 - Console testing, choice 3, \*other, and choice 4**

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 16 - Review of Enrollments.csv after console testing**

# Appendix B – Full code

# ------------------------------------------------------------------------------------------ #  
# Title: Assignment05  
# Desc: This assignment demonstrates using dictionaries, files, and exception handling  
# Change Log: (Who, When, What)  
# Brian Christopherson, 19-MAY-2025,Created Script  
# Brian Christopherson, 20-MAY-2025,Updated Script  
# ------------------------------------------------------------------------------------------ #  
  
# Define the Data Constants  
MENU: str = '''  
---- Course Registration Program ----  
 Select from the following menu:   
 1. Register a Student for a Course.  
 2. Show current data.   
 3. Save data to a file.  
 4. Exit the program.  
-----------------------------------------   
'''  
FILE\_NAME: str = "Enrollments.json"  
  
# Define the Data Variables and constants  
student\_first\_name: str = '' # Holds the first name of a student entered by the user.  
student\_last\_name: str = '' # Holds the last name of a student entered by the user.  
course\_name: str = '' # Holds the name of a course entered by the user.  
file = None # Holds a reference to an opened file.  
menu\_choice: str # Hold the choice made by the user.  
student\_data: dict = {} # one row of student data  
students: list = [] # a table of student data  
  
# Load the json module  
import json  
  
# When the program starts, read the file data into a list of lists (table)  
# Extract the data from the file  
try:  
 file = open(FILE\_NAME, "r")  
 students = json.load(file)  
 file.close()  
except FileNotFoundError as e:  
 print("File must exist before running this script!]\n")  
 print("Built-In Python error info: ")  
 print(e, e.\_\_doc\_\_, type(e), sep='\n')  
except Exception as e:  
 print("There was a non-specific error!\n")  
 print("-- Technical Error Message -- ")  
 print(e, e.\_\_doc\_\_, type(e), sep='\n')  
finally:  
 if file.closed == False:  
 file.close()  
  
# Present and Process the data  
while (True):  
  
 # Present the menu of choices  
 print(MENU)  
 menu\_choice = input("What would you like to do: ")  
  
  
 # Input user data  
 if menu\_choice == "1": # This will not work if it is an integer!  
 try:  
 student\_first\_name = input("Enter the student's first name: ")  
 if not student\_first\_name.isalpha():  
 raise ValueError("The first name should not contain numbers.")  
 student\_last\_name = input("Enter the student's last name: ")  
 if not student\_last\_name.isalpha():  
 raise ValueError("The last name should not contain numbers.")  
 course\_name = input("Please enter the name of the course: ")  
 student\_data = {"FirstName": student\_first\_name,  
 "LastName": student\_last\_name,  
 "CourseName": course\_name}  
 # Add new student to the students table  
 students.append(student\_data)  
 print(f"You have registered {student\_first\_name} {student\_last\_name} for {course\_name}.")  
 except ValueError as e:  
 print(e) # Prints the custom error message  
 print("--Technical Error Message-- ")  
 print(e.\_\_doc\_\_)  
 print(e.\_\_str\_\_())  
 except Exception as e:  
 print("There was a non-specific error!\n")  
 print("-- Technical Error Message-- ")  
 print(e.\_\_doc\_\_, type(e), sep="\n")  
 continue  
  
 # Present the current data  
 elif menu\_choice == "2":  
 # Process the data to create and display a custom message  
 print("-"\*50)  
 for student in students:  
 print(f"{student['FirstName']} {student['LastName']} is enrolled in "\  
 f"{student['CourseName']}")  
 print("-"\*50)  
 continue  
  
 # Save the data to a file  
 elif menu\_choice == "3":  
 try:  
 file = open(FILE\_NAME, "w")  
 json.dump(students, file, indent=2)  
 file.close()  
 print("The following data was saved to file!")  
 for student in students:  
 print(f"{student['FirstName']} {student['LastName']} is enrolled in " \  
 f"{student['CourseName']}")  
 except TypeError as e:  
 print("Please check that the data is a valid JSON format\n")  
 print("-- Technical Error Message -- ")  
 print(e, e.\_\_doc\_\_, type(e), sep='\n')  
 except Exception as e:  
 print("-- Technical Error Message -- ")  
 print("Built-In Python error info: ")  
 print(e, e.\_\_doc\_\_, type(e), sep='\n')  
 finally:  
 if file.closed == False:  
 file.close()  
 continue  
  
 # Stop the loop  
 elif menu\_choice == "4":  
 break # out of the loop  
 else:  
 print("Please only choose option 1, 2, 3, or 4")  
  
print("Program Ended")