

Alberto Arriola
February 26, 2025
IT FDN 100 A
Assignment 06

GitHub URL: <https://github.com/albertoarriola/IntroToProg-Python-Mod06>

Functions, Classes and Separation of Concerns

Intro

This week I learned about functions which are usable blocks of code that allow for modularity and reusability. By placing code in functions, I am able to access that code from the main body of my program whenever I need to. This is especially helpful for code that was previously repeated in earlier assignments. By putting repeated code into a function, if I need to make changes, I only have to change it in one place. This is particularly helpful in large programs with many lines of code.

I also learned about global and local variables, and when to use one versus the other. When working with functions, it is good to declare variables locally if they are only used within the function. If a variable will be used in other functions or in the main body of code, it is better to use a global variable.

After learning about global and local variables, and when to use them, I learned how to pass arguments to a function using parameters. Using parameters is cleaner than using global variables inside the function because it truly encapsulates what the function is trying to do. The function works with the passed in values, and then the results are returned to the main body of the code.

Functions can then be grouped under classes that contain functions with a common purpose. This is a great way to make the code more readable.

Finally, I learned about the concept of Separation of Concerns. This concept deals with grouping the classes according to the associated processes. For my assignment, I have a section for functions dealing with the external file under class FileProcessor. I have another section for functions dealing with input and output under class IO. This makes the program easier to read and understand because the functions are organized based on the purpose of the code.

Creating the Program

I started the program with a header section that includes the Title, Description and Change Log for the program.

```
# ----- #
# Title: Assignment06
# Desc: This lab demonstrates using input and output to a file, functions and classes
# Change Log: (Who, When, When)
#   Alberto Arriola, 2/28/2025, Created script
#   Alberto Arriola, 3/1/2025, Added comments
# ----- #
```

Fig 1.1: Script Header

Import

I used an Import statement to access the JSON modules that would allow my program to use the functions necessary to read to and write from the JSON file.

```
import json
```

Fig. 1.2 Import JSON statement.

Constants and Variables

I created two constants as specified in the acceptance criteria. The first constant, MENU, is a string with multiple lines so I used triple quotes to contain the text. The second constant, FILE_NAME, is a string that contains the name of the external JSON file that the program reads from and later writes to.

```
# 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"
```

Fig. 1.3 Constants used in Assignment06.py.

I created one variable of type String, one of type Object, and one of type List. For the List variable, I specified that it is a List of Dictionaries and that the Dictionaries contain three strings. Because I moved a lot of code into functions, I was able to remove them from the main code and declare them as local variables inside the appropriate functions.

```
# Variables
file: object = None
menu_choice: str = " "
students: list = [dict [str, str, str]]
```

Fig. 1.4 Variables used in Assignment06.py.

Classes

Right below the variable and import statement, I defined my classes. In accordance with the concept of Separation of Concerns, I divided my classes into processes dealing with the external file, and processes dealing with inputs and outputs.

FileProcessor class

The first class is titled FileProcessor. This class contains functions that read and write to the external file. A description of the class appears below the title.

```
# Define Classes

class FileProcessor: 2 usages
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    Alberto Arriola, 2/28/2025, Created Class
    """
```

Fig. 1.5 FileProcessor class containing functions accessing the external file.

The first function under the FileProcessor class is the read_data_from_file() function. A description of the function appears right below the title. This function reads data from the external json file. The name of the external file is passed to the function using the file_name parameter.

The student_list List variable is declared locally in the function. It is used to hold the data downloaded from the external json file.

A try-except block is used to catch any errors that may occur when trying to read data from the external file. The external file is opened using the open() function with the name of the external file and "r" for read being passed as parameters to the function. The result is assigned to the file Object variable.

There are two except statements to catch errors. The first one catches a FileNotFoundError. If this error occurs, the function calls the output_error_message function in the IO class and passes a custom error message and the error as parameters. The second one catches an other error that might occur and again call the output_error_message function in the IO class passing a custom error message and the error. If either error occurs, no information is read from the external file and an empty student_list is returned to the main program.

If no error occurs, the else statement is triggered and the contents of the external file are downloaded into the student_list List variable. This information is then returned to the main program and the user is asked to make another selection from the menu.

```
@staticmethod 1 usage
def read_data_from_file(file_name: str):
    """
    This function opens an external JSON file and reads the data
    into a List of Dictionaries.
    ChangeLog: (Who, When, What)
    Alberto Arriola, 2/28/2025, Created function
    :parameters: file_name: name of external JSON file
    :return: student_list: List of Dictionaries containing student information
    """
    student_list: list = []
    try:
        file = open(file_name, "r")
    except FileNotFoundError as e:
        IO.output_error_message( message: f"File \"{FILE_NAME}\" does not exist.", e)
    except Exception as e:
        IO.output_error_message( message: "An unexpected error occurred.", e)
    else:
        student_list = json.load(file)
        file.close()
    return student_list
```

Fig. 1.6 The read_data_from_file() function reads data from the external json file.

The second function found under the FileProcessor class is the write_data_to_file() function. A description of the function appears below the title. The name of the external file

is passed to the function as type String and assigned to `file_name`. The current student information is passed to the function and assigned to the `student_data` List.

It begins with an if-elif-else statement to handle situations where there is no data to write to the external file. This only occurs if there is no external json file to read information from and no new data has been entered by the user. The if statement is triggered if the `student_data` List variable is empty. If there is no data in `student_data`, then “not student_data” is evaluated as True and a message is printed letting the user know that “There is no student data to upload.”

If there is data in the `student_data` List variable, then “not student_data” is evaluated as False and the else statement is triggered. The external file is opened using the `open()` function and the `file_name` containing the name of the external file, and “w” for write arguments are passed to the `open()` function.

Again, a try-except block is used to catch errors that may occur during the attempt to write data to the external file. The program uses the `json dump()` function to write the information in `student_data` to the external file.

The first except statement attempts to catch a `NameError` error. If this occurs, the `output_error_message()` function in the IO class is called and a custom message and the exception information are sent as parameters.

The second except statement catches any other errors and also sends a custom message and the exception information to the `output_error_message()` function in the IO class.

If no errors occur, the external file is closed using the `close()` function and the information written to the external file is displayed. A nested for Loop iterates through all the students in the `student_data` List and prints out a message formatted using f string telling the user the student’s first name, last name and the name of the course they registered for. An extra blank line is printed using the `print()` function just to make things look pretty, and the user is returned to the main code to make another menu selection.

```

@staticmethod 1 usage
def write_data_to_file(file_name: str, student_data: list):
    """
    This function opens an external JSON file and writes the student
    registration data to the file.
    ChangeLog: (Who, When, What)
    Alberto Arriola, 2/28/2025, Created function
    :parameters: file_name: name of external JSON file,
    student_data: List of Dictionaries containing student information
    :return: None
    """

    if not student_data:
        print("There is no student data to upload.")
    else:
        file = open(file_name, "w")
        try:
            json.dump(student_data, file)
        except NameError as e:
            IO.output_error_message(message="An error occurred because the file name for the write function is incorrect.", e)
        except Exception as e:
            IO.output_error_message(message="An unexpected error occurred.", e)

        file.close()

        for student in student_data:
            print(f"{student['FirstName']} {student['LastName']} has "
                  f"been registered for {student['CourseName']}".)

    print()

```

Fig. 1.7 The write_data_from_file() function writes data to the external json file.

IO class

Below the write_data_from_file() function is the IO class. This class contains functions that either accept input from the user or output data. A description of the class appears below the title.

```

class IO: 12 usages
    """
    A collection of presentation layer functions that manage user input and output
    ChangeLog: (Who, When, What)
    Alberto Arriola, 2/28/2025, Created Class
    """

```

Fig. 1.8 The IO class containing functions that deal with user

The first function in the class IO section is the output_error_message() function. A description of the function appears below the title. This function prints out error information if one occurs while the program is running. A message string and the error information are passed to the function as parameters.

The message String variable contains a custom message which is printed for the user. A nested if statement determines if an error parameter has been passed to the function. If one has been passed, then the passed error information is printed out. If an error parameter has not been passed to the function, then the print() functions in the if

statement are skipped. The user is then returned to the code that called the `output_error_message()` function.

```
@staticmethod 8 usages
def output_error_message(message: str, error: Exception = None):
    """
    This function displays error messages to the user
    ChangeLog: (Who, When, What)
    Alberto Arriola, 2/28/2025, Created function
    : parameters: message = error message in the form of a string,
    error = Exception
    : return: None
    """
    print(message, end="\n")
    if error is not None:
        print("-- Technical Error Message -- ")
        print(error, error.__doc__, type(error), sep='\n')
    print()
```

Fig. 1.9 The `output_error_message()` function prints out error messages for the user.

The next function is the `output_menu()` function. A description of the function appears below the title. The value of the String containing the menu is passed to the function in the `menu_display` parameter. The string is then displayed using the `print()` function.

```
@staticmethod 1 usage
def output_menu(menu_display: str):
    """
    This function prints out the selection Menu.
    ChangeLog: (Who, When, What)
    Alberto Arriola, 2/28/2025, Created function
    :parameters: menu_display: menu String displaying menu selections
    :return: None
    """
    print(menu_display)
```

Fig. 1.10 The `output_menu()` function displays the menu.

The next function is the `input_menu_choice()` function. A description of the function appears below the title. The string that contains the menu information is passed to the function in the `menu_display` String parameter.

A while Loop is used to make sure the user makes a valid selection. After an empty line is printed out using the `print()` function, a call is made to the `output_menu()` function in the IO class and the value of the `menu_display` Str is passed to the function. The `output_menu()` function prints out the menu for the user. Then an input statement is used ask the user to make a selection. The result is stored in the `user_selection` String variable.

A nested if statement checks if the value of `user_selecion` is “1”, “2”, “3”, or “4”. If it is not, then a message asking the user to select “1”, “2”, “3”, or “4” is displayed. A `continue` statement returns the user to the start of the While True Loop and the user is again asked to make a selection from the menu.

If a valid selection is made, then the `else` statement is triggered and the user escapes the while Loop using the `break` statement. The value in `user_selection` is then returned to the main code.


```

@staticmethod 1 usage
def input_menu_choice(menu_display: str):
    """
    This function retrieves the user's menu selection.
    ChangeLog: (Who, When, What)
    Alberto Arriola, 2/28/2025, Created function
    :parameters: menu_display = menu String from constant MENU
    :return: user_selection: String containing user's menu selection
    """
    user_selection: str = ""

    while True:
        print()
        IO.output_menu(menu_display)
        user_selection = input("Please make a selection: ")
        if user_selection not in ("1,2,3,4"):
            print("Please select 1, 2, 3 or 4.")
            continue
        else:
            break
    print()
    return user_selection

```

Fig. 1.11 The `input_menu_choice()` function requests and records the menu selection from the user.

The next function is the `input_student_data()` function. The description of the function appears below the title.

```

@staticmethod 1 usage
def input_student_data(student_list: list):
    """
    This function asks the user to enter student information and
    appends it to the existing list of student information.
    ChangeLog: (Who, When, What)
    Alberto Arriola, 2/28/2025, Created function
    :parameters: student_list: List of Dictionaries containing student data
    :return: student_list: List of Dictionaries containing student data
    """

```

Fig. 1.12 The `input_student_data()` function is described below the title.

A list of local variables is declared just below the function description. A while Loop begins containing the steps to have the user input the student's first name. It contains a try-except block that first uses the `input()` function to ask the user to enter the student's first name. The results is assigned to the local `student_first_name` variable. Then a nested if statement uses the `isalpha()` function to determine if there are any non-alpha characters in `student_first_name`. If there are any non-alpha characters, then a custom `ValueError` is raised and an except statement calls the `output_error_message()` function in the IO class, passing a custom error message string, but no other error information. The `output_error_message()` function prints out the error message which tells the user, "The student's first name should only contain letters." A second except statement catches any other errors that may occur. If an error other than the custom `ValueError` occurs, the `output_error_message()` function is called and an error message and Python's error message are sent as parameters.

If any errors occur, they are handled, and then the continue statement returns the user to the beginning of the while Loop. The user is asked again to enter the student's first name. The while Loop will continue until the user enters a first name with only alpha characters and no errors occur. Then the else statement is triggered and the break statement breaks the user out of the loop.

The code to get the student's last name from the user is similar. It occurs in a while Loop and checks for alpha only characters as well as any other errors. Non-alpha characters cause the `ValueError` calling the `output_error_message()` function. Again, any errors lead to the continue statement that keeps the user in the while Loop entering the student's last name. Once the user enters a valid last name without any errors occurring, the break statement is triggered and the user exits the while Loop.

```

# variables local to the function
student_first_name: str = " "
student_last_name: str = " "
course_name: str = " "
student_data: dict[str, str] = {}

while True:
    try:
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError()
    except ValueError as e:
        IO.output_error_message("The student's first name should only contain letters.")
    except Exception as e:
        IO.output_error_message( message: "An unexpected error occurred.", e)
        continue
    else:
        break

while True:
    try:
        student_last_name = input("Enter the student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError()
    except ValueError as e:
        IO.output_error_message("The student's last name should only contain letters.")
    except Exception as e:
        IO.output_error_message( message: "An unexpected error occurred.", e)
        continue
    else:
        break

```

Fig. 1.13 The `input_student_data()` function collects a student's first name and last and throws an error if either one contains non-alpha characters.

Finally, the `input()` function asks the user to enter the student's course name and assigns it to the local `course_name` String variable. The `student_first_name`, `student_last_name`, and `course_name` values are then assigned to the local `student_data` Dictionary variable with the Keys "FirstName", "LastName", and "CourseName" respectively. The `append()` function is used to add the `student_data` Dictionary value to the `student_list` List that was passed to the function as a parameter. The value of the new `student_list` is then returned to the main program.

```
course_name = input("Enter the course name: ")
student_data = {"FirstName": student_first_name,
                "LastName": student_last_name,
                "CourseName": course_name}
student_list.append(student_data)
print()
return student_list
```

Fig. 1.14 The `input_student_data()` function collects the course name from the user, stores all the student information in the `student_data` Dictionary variable, and appends it to the `student_list` List.

The final function in the IO class is the `output_student_courses()` function. This function is used to print out all the student information in the `students` List. The description appears below the title.

The list of student information is passed to the function and assigned to `student_info`. An if statement checks if there is any data in the `student_info` List. If there is no information, then a message is printed informing the user that, "There is no student data to display."

If `student_info` is not empty, the else statement is triggered and an empty line is printed followed by a header that reads: "The current data is: ". A nested for Loop runs through all the student entries in the `student_info` List and prints out the student's first name, last name, and course name accessing the information in the Dictionary entry using Key Values. After all the students' information are printed out the function ends and returns to the main code without returning anything.

```

@staticmethod 1 usage
def output_student_courses(student_info: list):
    """
    This function displays current student information.
    ChangeLog: (Who, When, What)
    Alberto Arriola, 2/28/2025, Created function
    :parameters student_info: List of Dictionaries containing student data
    :return: None
    """
    if not student_info:
        print("There is no student data to display.")
    else:
        print()
        print("The current data is: ")
        for student in student_info:
            print(student["FirstName"], student["LastName"]
                  + ", " + student["CourseName"])
        print()

```

Fig. 1.15 The `output_student_courses()` function displays the current list of student information.

Main Code

The first thing the main code does is call the `read_data_from_file()` function from the `FileProcessor` class, passing in the constant `FILE_NAME`. The results read from the external JSON file and returned by the `read_data_from_file()` function are assigned to the `students` List variable.

Then a while Loop begins so that the program will continue to run until the user enters “4” in the menu to exit the program. The `input_menu_choice()` function in the `FileProcessor` class is called and the `MENU` constant is passed as a parameter. This prints out the menu and asks the user to input a selection. The user’s choice is returned and assigned to the `menu_choice` String variable.

Then an if-elif-else statement is used to perform the appropriate action based on the user’s input. If the user enters “1”, then the `input_student_data()` function in the `IO` class is called. The `students` List is passed as a parameter, and after the function adds new student information, the new list is passed back and assigned to the `students` List.

If the user enters “2”, then the `output_student_courses()` function in the `IO` class is called. The `students` List is passed as a parameter, and the data is displayed. No information is returned by the function.

If the user enters “3”, then the `write_data_to_file()` function in the `FileProcessor` class is called. The `FILE_NAME` constant and the `students` List are passed to the function and the data in the `students` List is written to the external JSON file. No information is returned by the function.

And finally, if the user enters “4”, the message, “Goodbye!” is displayed, the `break` statement is used to exit the `while` Loop, and the program ends.

```
# open and read from json file
students = FileProcessor.read_data_from_file(FILE_NAME)

while (True):

    menu_choice = IO.input_menu_choice(MENU)

    # Selection 1. Register a student for a course
    if (menu_choice == "1"):
        students = IO.input_student_data(students)
    # Selection 2. Show current data
    elif (menu_choice == "2"):
        IO.output_student_courses(students)
    # Selection 3. Save data to a file
    elif (menu_choice == "3"):
        FileProcessor.write_data_to_file(FILE_NAME, students)
    # Selection 4. Exit the program
    elif (menu_choice == "4"):
        print("Goodbye!")
        break
```

Fig. 1.16 The main code section of the program calls specific functions based on the desired action.

Comments

Finally, I added comments to each line of code using the '#'. This will make it easier for other developers to figure out what I was trying to do with each line.

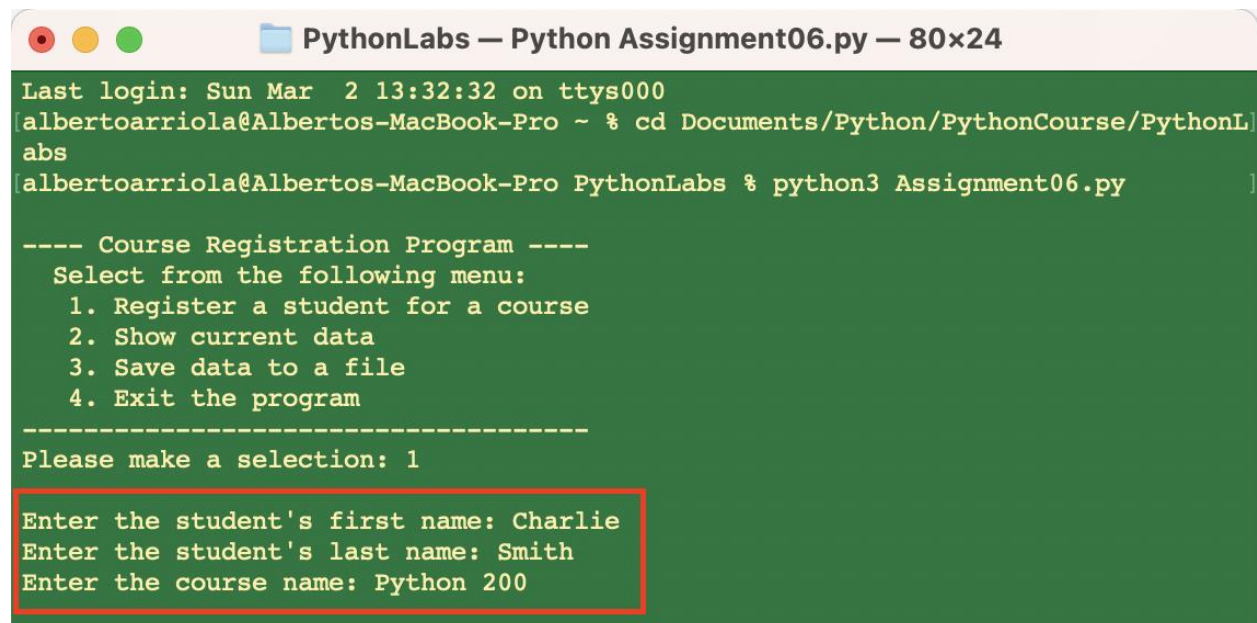
Testing the Program

Now that the scripting is complete, it is time to test the program. I ran it first using Terminal on my mac. I opened the Terminal app and then used the 'cd' command to point to the directory in which my script was saved.

I typed 'python3 Assignment06.py' to run the script.

Clean Run

I entered "1" to select "Register a student for a course". I was prompted for the student's first name and entered "Charlie". Then I was prompted for the student's last name and entered "Smith". Finally, I was prompted for the course name and entered "Python 200". After entering the data, I was returned to the Menu.



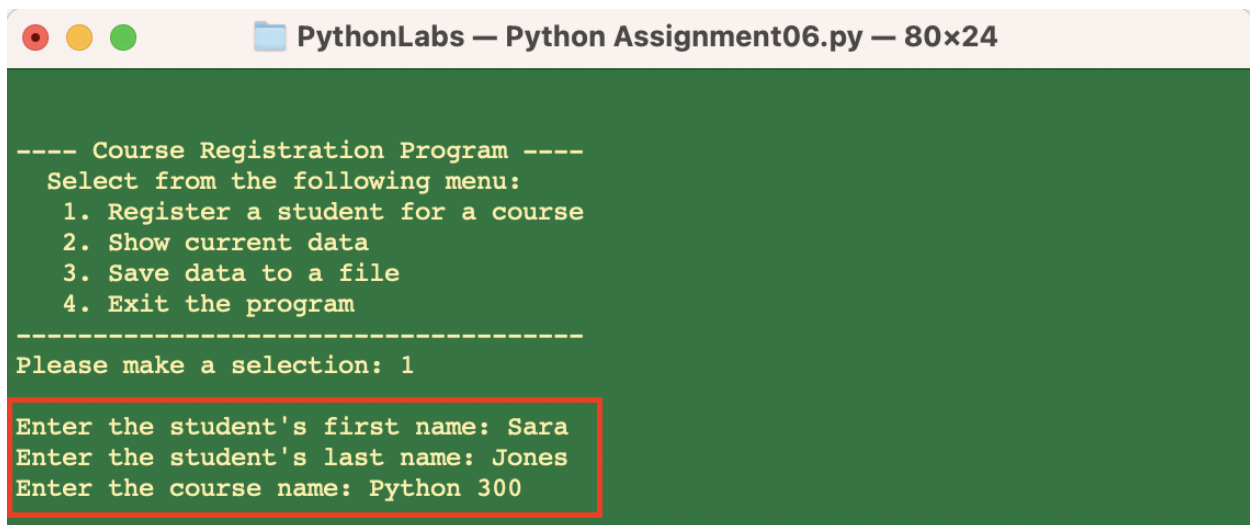
```
Last login: Sun Mar  2 13:32:32 on ttys000
[albertoarriola@Albertos-MacBook-Pro ~ % cd Documents/Python/PythonCourse/PythonLabs
[albertoarriola@Albertos-MacBook-Pro PythonLabs % python3 Assignment06.py

---- 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
-----
Please make a selection: 1

Enter the student's first name: Charlie
Enter the student's last name: Smith
Enter the course name: Python 200
```

Fig. 1.17 Selecting "1 – Register a student for a course".

I entered "1" again to add a second student. This time, I entered "Sara Jones" for the class "Python 300".



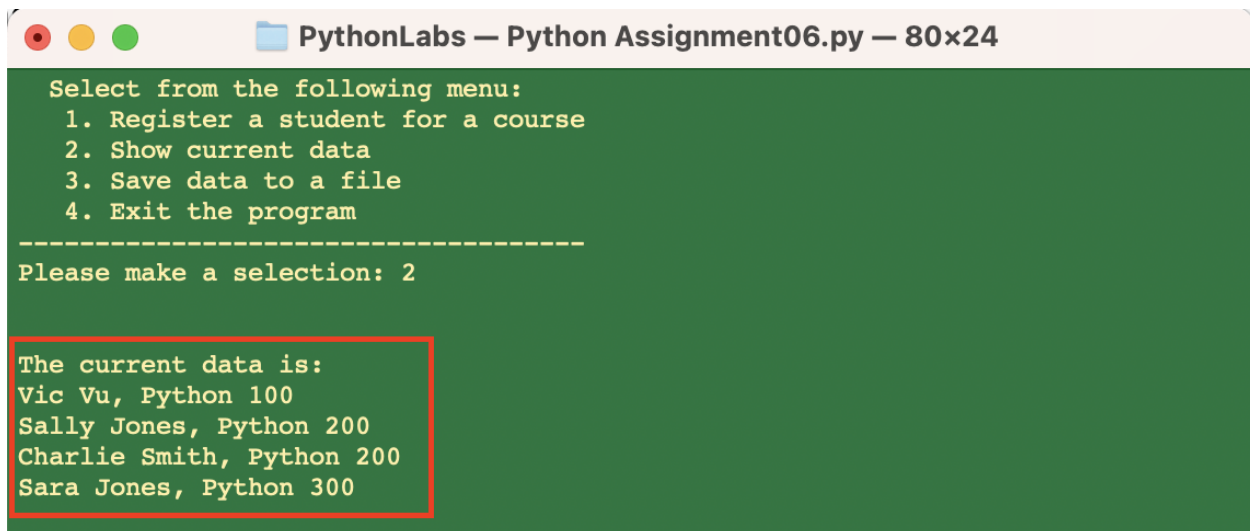
```
PythonLabs — Python Assignment06.py — 80x24

---- 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
-----
Please make a selection: 1

Enter the student's first name: Sara
Enter the student's last name: Jones
Enter the course name: Python 300
```

Fig. 1.18 Selecting “1 – Register a student for a course” and entering a second student.

I entered “2” to select “Show current data” and the program printed out the first name, last name, and the course name for the two students that were already in the external Enrollments.json file, as well as the two students that I added. Then it returned me to the Menu.



```
PythonLabs — Python Assignment06.py — 80x24

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
-----
Please make a selection: 2

The current data is:
Vic Vu, Python 100
Sally Jones, Python 200
Charlie Smith, Python 200
Sara Jones, Python 300
```

Fig. 1.19 Selecting “2 – Show current data”.

At the next Menu prompt, I entered “3” to select “Save data to a file”. The student information was written to the external Enrollments.json file. Then the program printed out the line “[student first name] [student last name] has been registered for [course name].” for each of the students in the file. Then it returned me to the Menu.


```
---- 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
-----
Please make a selection: 3

Vic Vu has been registered for Python 100.
Sally Jones has been registered for Python 200.
Charlie Smith has been registered for Python 200.
Sara Jones has been registered for Python 300.
```

Fig. 1.20 Selecting “3 – Save data to file”.

I opened the Enrollments.json file in a text editor and verified that all four students appeared in the file as Dictionaries in a List.

Name	Date Modified	Size	Kind
Enrollments.json	Today at 1:40 PM	288 bytes	JSON
Assignment06.py	Today at 1:18 PM	21 KB	Python Script

Fig. 1.21 Enrollments.json file in PythonLabs folder.

```
[{"FirstName": "Vic", "LastName": "Vu", "CourseName": "Python 100"}, {"FirstName": "Sally", "LastName": "Jones", "CourseName": "Python 200"}, {"FirstName": "Charlie", "LastName": "Smith", "CourseName": "Python 200"}, {"FirstName": "Sara", "LastName": "Jones", "CourseName": "Python 300"}]
```

Fig. 1.22 Data in the Enrollments.json file.

Finally, I entered “4” to select “Exit the program”. The program printed the line “Goodbye” and ends.

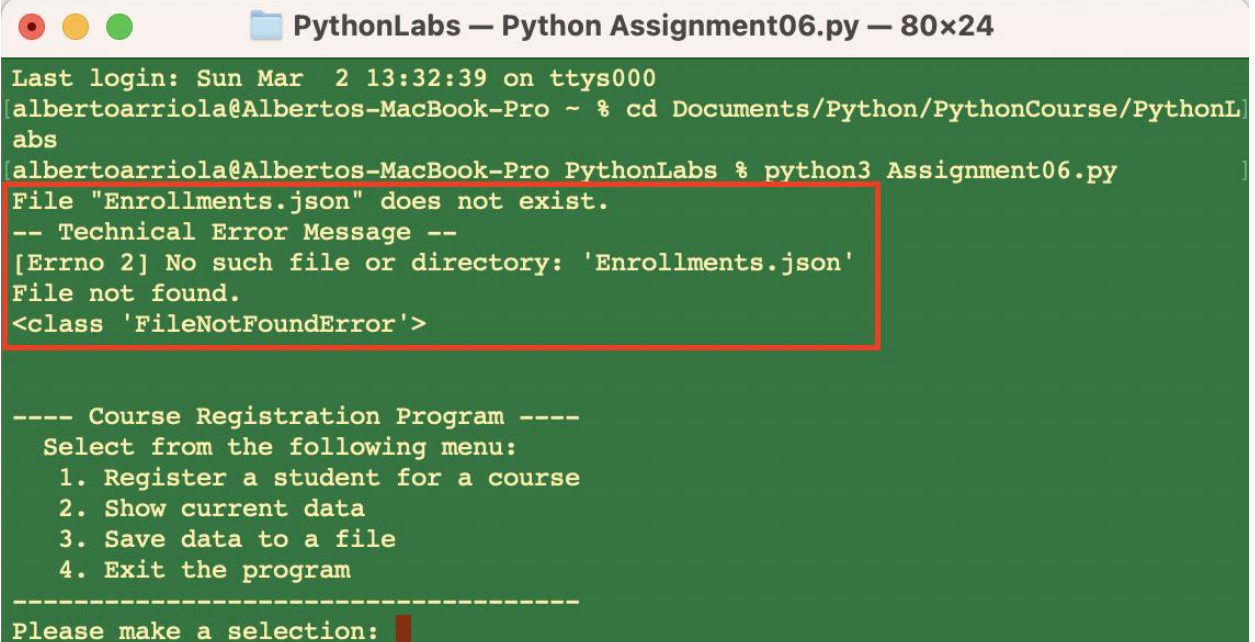
```
---- Course Registration Program ----
Select from the following menu:
1. Register a student for a course
2. Show current data
3. Save data to a file
4. Exit the program
-----
Please make a selection: 4

Goodbye!
albertoarriola@Albertos-MacBook-Pro PythonLabs %
```

Fig. 1.23 Selecting “4 – Exit the program”.

Error Handling Run

I decided to do a run that tested my error handling. I started by deleting the Enrollments.json file. When I started the program, the FileNotFoundError occurred. The custom error message was printed as well as the Python error messages.



```
PythonLabs — Python Assignment06.py — 80x24
Last login: Sun Mar  2 13:32:39 on ttys000
albertoarriola@Albertos-MacBook-Pro ~ % cd Documents/Python/PythonCourse/PythonL
abs
albertoarriola@Albertos-MacBook-Pro PythonLabs % python3 Assignment06.py
File "Enrollments.json" does not exist.
-- Technical Error Message --
[Errno 2] No such file or directory: 'Enrollments.json'
File not found.
<class 'FileNotFoundError'>

---- 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
-----
Please make a selection: 
```

Fig. 1.24 Error message is received when the external Enrollments.json file does not exist in the current directory.

Then I selected “1”. I entered a first name that contained a number and verified that a custom error message was printed out. The user was given another chance to enter a student’s first name. I also verified that the error was received if the user tried to enter a symbol as part of the first name. The program finally moved on after an name with only alpha characters was entered.

```
---- 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
```

```
-----  
Please make a selection: 1
```

```
Enter the student's first name: Jo3  
The student's first name should only contain letters.
```

```
Enter the student's first name: Jo%  
The student's first name should only contain letters.
```

```
Enter the student's first name: █
```

Fig. 1.25 Error handling if user enters a non-alpha character in the student's first name.

I also verified that the same error handling occurred when the user entered the student's last name.

```
Enter the student's first name: Victor  
Enter the student's last name: 7on  
The student's last name should only contain letters.
```

```
Enter the student's last name: $ammy  
The student's last name should only contain letters.
```

```
Enter the student's last name: █
```

Fig. 1.26 Error handling if user enters a non-alpha character in the student's last name.

The program continued after an acceptable first name and last name were entered.

Finally, I entered an invalid selection at the menu and verified that the user was told to only enter "1", "2", "3", or "4". The menu was re-displayed and the user was asked to make a selection again. This ran until a valid selection was made.

```
PythonLabs — Python Assignment06.py — 80x24

3. Save data to a file
4. Exit the program
-----
Please make a selection: 5
Please select 1, 2, 3 or 4.

---- 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
-----
Please make a selection: %
Please select 1, 2, 3 or 4.

---- 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
-----
Please make a selection: █
```

Fig. 1.27 Error message is received if the user enters a selection other than “1”, “2”, “3”, or “4”.

Finally, I entered “4” to select “Exit the program”. The program printed the line “Goodbye” and ended.

PyCharm test

I also ran the script in PyCharm and verified that each of the selections behaved correctly. I deleted the Enrollments.csv file to verify again that program creates it if it doesn’t exist.

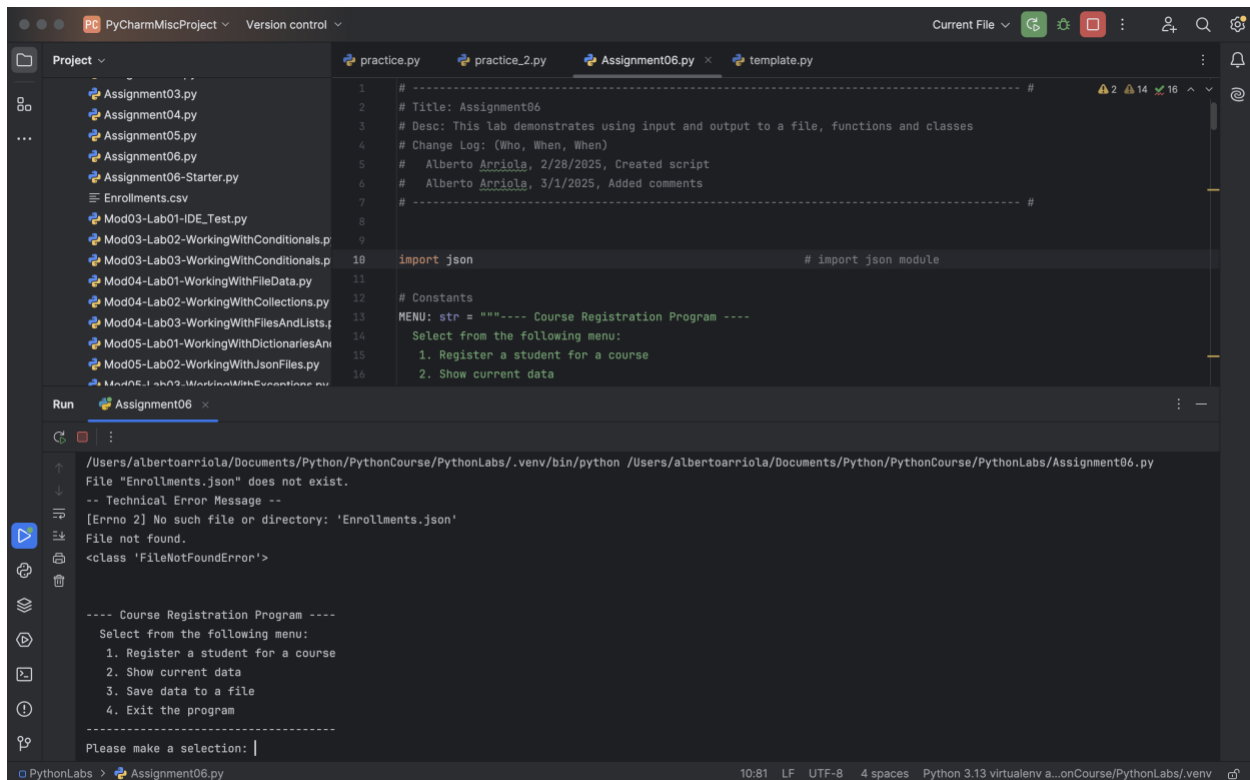


Fig. 1.28 Execution of the script in PyCharm.

After adding students, I opened the Enrollments.json file in PyCharm to verify that the student data was successfully written to the external file.

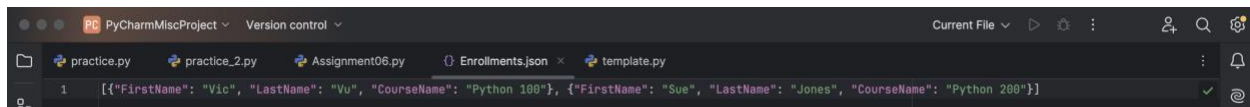


Fig. 1.29 The uploaded data is written to the Enrollments.json file.

GitHub

I created a new repository in GitHub for Module 6 and uploaded my assignment, the external JSON file, and my knowledge document. Here is the link to my project:

<https://github.com/albertoarriola/IntroToProg-Python-Mod06> .

Summary

Module 6 was all about making the code more modularized and easier to read. To do this, I put code blocks into functions. Then I put the functions into classes. This allowed me to call the functions whenever I needed to perform a specific process.

By putting code blocks into functions, I was able to access them from anywhere in the program. I also used local variables inside the functions to make sure they were entirely encapsulated and not reliant on variables found in the main code.

Following the concept of Separation of Concerns, I grouped my functions into two classes that had common purposes. The FileProcessor class contains functions that interact with the external json file. This includes the read_data_from_file() function, and the write_data_to_file() function.

The IO class contains functions that either receive input, produce output, or both. There are five functions in the IO class, output_error_message(), output_menu(), input_menu_choice(), input_student_data(), and output_student_courses().

By using the classes and functions, the main code of the program becomes easier to read, reduced to only about 16 lines. I suspect that the usefulness of this type of code organization will only become more obvious as my program grows.