

Alberto Arriola
March 10, 2025
IT FDN 100 A
Assignment 07

GitHub URL: <https://github.com/albertoarriola/IntroToProg-Python-Mod07>

Classes, Functions and GitHub

Intro

This week I learned more about classes, functions and statements. Statements can be grouped together to create a function. This allows the user to access frequently used lines of code easily without having to rewrite the statements whenever necessary. Common functions can be grouped together into classes. Using classes, the code can be organized according to the concept of separation of concerns. The code becomes easier to read since it is structured in a way that makes sense.

Classes are designed to focus on data, processing or presentation. I used all three types of classes in my assignment and organized my code accordingly. The data class has special methods used for creating an object of that class type. Attributes are passed to the constructor of the class and an instance of the class is created. Getter and setter property functions for the data class allow the user to access data as well as perform actions on the data like validation and error handling. Using constructors, getters and setters allows for encapsulation, meaning that everything needed to create an instance of the class occurs within the class.

I also learned about inheritance between parent and a child classes. A child class can inherit data and behaviors from a parent class. This allows a developer to create a general class that can be added to in order build more specified objects. I used this concept in my program by creating a Person class and then creating a Student Sub class. A Student object has the first_name and last_name properties of the Person class, but it also has a course_name property.

Finally, I learned about accessing GitHub locally in PyCharm using Git rather than uploading directly to the website. Git allows a developer to clone a project and work on a separate branch based on the main code. The developer can make changes to the branch and then commit and merge the changes back to the main project in GitHub. Git tracks and keeps a history of the changes so that there is a record of who made what changes and when.

Creating the Program

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

```
# -----
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
#   Alberto Arriola, 3/9/2025, Created Script
#   Alberto Arriola, 3/9/2025, Added code to finally statement to handle file = None
#   Alberto Arriola, 3/9/2025, Added if statements to handle no student data to display
#                                     or upload
# -----
```

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.

```
# 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"
```

Fig. 1.3 Constants used in Assignment06.py.

I created one variable of type List, and one of type String. I was able to remove variables that were in the main body of previous iterations of my code and declare them as local variables inside functions.

```
# Define the Data Variables
students: list = [] # list variable
menu_choice: str # str variable
```

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 data, processing and presentation.

Person class

The first class is titled Person. This class contains methods that create a Person object with first_name and last_name variables. The __init__ method is a constructor that allows the code to initialize an instance of the Person class with first_name and last_name attributes.

```
class Person: 1 usage  👤 albertoarriola
    """Person class..."""
    # constructor for Person instance
    def __init__(self, first_name: str = "", last_name: str = ""):
        """This method is the Person class constructor..."""
        self._first_name = first_name # first_name property assign
        self._last_name = last_name # last_name property assign
```

Fig. 1.5 Person class with init method.

I included getter methods in the Person class to handle when the code wants to access the first_name or last name attributes of a Person object. Both of them return the a value in the title() format meaning that the first letter is capitalized.

```
# getter for the first_name property
@property 6 usages (2 dynamic)  ⚡ albertoarriola
def first_name(self):
    """This method is the first_name ge
    return self._first_name.title()
```

Fig. 1.6 Getter method for `first_name` property in `Person` class.

```
# getter for the last_name property
@property 6 usages (2 dynamic)  ⚡ albertoarriola
def last_name(self):
    """This method is the last_name gette
    return self._last_name.title() #
```

Fig. 1.7 Getter method for `last_name` property in `Person` class.

I included setter methods in the `Person` class to handle when the code wants to modify the `first_name` or `last_name` attributes of a `Person` object. The setter methods take a value that is passed to the method and assigns it to the `first_name` or `last_name` attribute. I added try-except blocks to perform error handling for when a value is assigned to the attribute. A custom exception uses the `isalpha()` function to make sure that the `first_name` and `last_name` attributes only contain letters. Encapsulation of the `Person` class is promoted by placing the error handling in the setter method.

```
# setter for the first_name property
@first_name.setter 5 usages (2 dynamic)  ⚡ albertoarriola
def first_name(self, value: str):
    """This method is the first_name setter for the Person class..."""
    # try-except block to handle ValueError
    try:
        if value.isalpha(): # check if first_name string contains only letters
            self._first_name = value # first_name string is set to passed value if it
        else:
            raise ValueError() # ValueError is raised if first_name doesn't contain on
    except ValueError as e:
        # Call to output_error_messages() function in IO class
        IO.output_error_messages("The student's first name should only contain letters.")
    except Exception as e:
        # Call to out_put_error_messages() function in IO class
        IO.output_error_messages( message: "An unexpected error occurred.", e)
```

Fig. 1.8 Setter method for `first_name` property in `Person` class.

```

# setter for the last_name property
@last_name.setter 5 usages (2 dynamic)  ⚡ albertoarriola
def last_name(self, value: str):
    """This method is the last_name setter for the Person class..."""
    # try-except block to handle ValueError
    try:
        if value.isalpha():      # check if last_name string contains only letters
            self._last_name = value    # last_name string is set to passed value if it co
        else:
            raise ValueError()      # ValueError is raised if first_name doesn't contain o
    except ValueError as e:
        # Call to output_error_messages() function in IO class
        IO.output_error_messages("The student's last name should only contain letters.")
    except Exception as e:
        # Call to out_put_error_messages() function in IO class
        IO.output_error_messages(message: "An unexpected error occurred.", e)

```

Fig. 1.9 Setter method for last_name property in Person class.

I also included `__str__` method in the Person class to override the default `__str__` method that determines how the Person object is displayed.

```

# override __str__() method to return Person data
def __str__(self):  ⚡ albertoarriola
    """This method is the __str__() override for the
    return f"{self._first_name}, {self._last_name}"

```

Fig. 1.10 `__str__` method in Person class to override how a Person object is displayed.

Student class

I then created the Student class which is a subclass of Person. This means that it inherits the first_name and last_name attributes from the Person class. The Student class adds the course_name attribute. It also includes an `__init__` method as a constructor that allows the code to initialize an instance of the Student class with first_name, last_name and course_name attributes.

```

class Student(Person): 3 usages  ⚡ albertoarriola
    """Student class that inherits from the Person class..."""
    # Student constructor that calls to the Person constructor for first_name and last_name
    def __init__(self, first_name: str = "", last_name: str = "", course_name: str = ""):
        """This method is the Person class constructor..."""
        super().__init__(first_name = first_name, last_name = last_name)    # call to Pers
        self._course_name = course_name    # course_name property assignment

```

Fig. 1.11 Student class is a subclass of the Person class with an init method.

The Student class includes getter, setter and __str__ methods for the course_name attribute.

```
# getter for course_name property
@property 4 usages (2 dynamic)  albertoarriola
def course_name(self):
    """This method is the last_name getter for the Person class..."""
    return self._course_name.title()    # return course_name string in Title Case

# setter for course_name property
@course_name.setter 3 usages (2 dynamic)  albertoarriola
def course_name(self, value: str):
    """This method is the last_name setter for the Person class..."""
    self._course_name = value    # course_name string is set to passed value

# override __str__() method to return Student data
def __str__(self):  albertoarriola
    """This method is the __str__() override for the Person class..."""
    return f"{self._first_name}, {self._last_name}, {self._course_name}"
```

Fig. 1.12 Getter, setter and __str__ methods for the course_name attribute in the Student class.

FileProcessor class

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

```
# Processing ----- #
class FileProcessor: 2 usages  albertoarriola
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    Alberto Arriola, 3/9/2025, Created Class
    """
```

Fig. 1.13 FileProcessor class containing functions accessing the external file.

The first function under the FileProcessor class is the read_data_from_file() function. 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.

```

# Processing ----- #
class FileProcessor: 2 usages 1 albertoarriola *
    """A collection of processing layer functions that work with Json files..."""
    @staticmethod 1 usage 1 albertoarriola *
    def read_data_from_file(file_name: str, student_data: list):
        """This function reads data from a json file and loads it into a list of dictionary rows..."""
        file: object = None
        list_of_dictionary_data: list[dict] = [] # declare local list variable
        # try-except block to handle error reading the external file
        try:
            file = open(file_name, "r") # open external json file
        except Exception as e:
            # call to output_error_messages() function in IO class
            IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
        else:
            student_json = json.load(file) # load contents of external file into student_json list
            file.close() # close the external file
            # for loop converts students of type Dictionary to Student type
            for student in student_json:
                student_object: Student = Student(first_name=student["FirstName"],
                                                    last_name=student["LastName"],
                                                    course_name=student[
                                                        "CourseName"]) # convert student Dictionary to Student
                student_data.append(student_object) # append student_object to student_data list of Students
            # finally statement to make sure the external file is closed
            finally:
                if file is None: # if statement triggered if file is set to None
                    file = open(file_name, "w") # creates a new external Enrollments.json file
                    json.dump(list_of_dictionary_data, file) # load list_of_dictionary_data to external file using .dump()
                    file.close()
                else:
                    file.close() # close external file if it is still open
            return student_data # return student_data list of Students to the main body

```

Fig. 1.14 The `read_data_from_file()` function reads data from the external json file.

The file object 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 is a single statement to catch any error. If an 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. If an 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_json` List variable. A for loop is used to convert the contents of the list from Dictionaries into Student objects and append the results to the `student_data` List.

A finally statement is used to make sure that the external file is closed.

The student_data list is then returned to the main program and the user is asked to make another selection from the menu.

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.

```
@staticmethod 1 usage 2 albertoarriola *
def write_data_to_file(file_name: str, student_data: list):
    """This function writes data to a json file with data from a list of dictionary rows..."""
    list_of_dictionary_data: list[dict] = [] # declare local list variable
    if len(student_data) == 0: # if statement triggered if student_data is empty
        print("There is no student data to upload.") # print message if there is no data to upload
    else: # else statement triggered if there is data in student_data
        # try-except block to handle errors writing to the external file
        try:
            file = open(file_name, "w") # open external file
            # for loop to convert students from Student to Dictionary type
            for student in student_data:
                student_json: dict = {"FirstName": student.first_name, "LastName": student.last_name, "CourseName": student.course_name}
                list_of_dictionary_data.append(student_json)
            json.dump(list_of_dictionary_data, file) # load list_of_dictionary_data to external file using .dump()
            file.close() # close external file
            # call to output_student_and_course_names() function in IO class
            IO.output_student_and_course_names(student_data=student_data)
        except Exception as e:
            message = "Error: There was a problem with writing to the file.\n"
            message += "Please check that the file is not open by another program."
            IO.output_error_messages(message=message, error=e)
        # finally statement to make sure the external file is closed
        finally:
            if file.closed == False: # determine if external file is closed
                file.close() # close external file if it is still open
```

Fig. 1.15 The write_data_from_file() function writes data to the external json file.

A local List variable called list_of_dictionary_data is declared. Then an if-else statement is used to handle situations where there is no data to write to the external file. This can occur if there is no external json file to read information from or the file exists but is empty. The if statement is triggered if the student_data List variable is empty. If there is no data in student_data, then 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 a try-except block is used to catch errors that may occur during the attempt to write data to the external file. A for Loop is used to iterate through all the students in the student_data List and convert the Student objects into Dictionaries. The results are appended to the list_of_dictionary_data List. The program uses the json dump() function to write the information in list_of_dictionary_data to the external file. 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. Then a call is made to the output_student_and_course_names() function and the student information is displayed.

If an error occurs, the `output_error_message()` function in the IO class is called and a custom message and the exception information are sent as parameters.

A finally statement determines if the external file has been closed. If the file has not been closed, then the `close()` function is used to close it.

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.

```
# Presentation ----- #
class IO: 12 usages  ⤴ albertoarriola *
    """
    A collection of presentation layer functions that manage user input and output

    ChangeLog: (Who, When, What)
    Alberto Arriola, 3/9/2025, Created Class

    """
```

Fig. 1.16 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.

```
@staticmethod 7 usages  ⤴ albertoarriola
def output_error_messages(message: str, error: Exception = None):
    """
    This function displays error messages to the user

    : param message: error message in the form of a string,
    : param error: Python Exception info
    : return: None

    ChangeLog: (Who, When, What)
    Alberto Arriola, 3/9/2025, Created function
    """
    print(message, end="\n\n")      # print out error message
    if error is not None:          # determine if Python error information
        print("-- Technical Error Message -- ")
        print(error, error.__doc__, type(error), sep='\n')
```

Fig. 1.17 The `output_error_message()` function prints out error messages for the user.

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.

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  albertoarriola
def output_menu(menu: str):
    """ This function displays the menu of choices to the user

    :param menu: string with menu text to display

    :return: None

    ChangeLog: (Who, When, What)
    Alberto Arriola, 3/9/2025, Created function
    """
    print(menu)      # print out menu
    print()          # extra printline
```

Fig. 1.18 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.

```

@staticmethod 1 usage  ⤴ albertoarriola
def input_menu_choice():
    """ This function gets a menu choice from the user

    :return: string with the users choice

    ChangeLog: (Who, When, What)
    Alberto Arriola, 3/9/2025, Created function
    """
    choice: str = ""    # declare local str variable
    # try-except block to handle invalid entry
    try:
        choice = input("Enter your menu choice number: ")    # print
        if choice not in ("1","2","3","4"):    # verify choice is valid
            raise Exception("Please, choose only 1, 2, 3, or 4.")
    except Exception as e:
        # call to output_error_messages() function in IO class
        IO.output_error_messages(e.__str__())    # Not passing e to
    return choice    # return choice string value to main body

```

Fig. 1.19 The `input_menu_choice()` function requests and records the menu selection from the user.

The string variable `choice` is declared locally. A try-except block is used to handle if the user enters anything other than “1”, “2”, “3”, or “4”. If anything else is entered, a custom Exception is raised and sent to the `output_error_messages()` function in the IO class. The choice is then returned to the main body of the program.

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

```

@staticmethod 2 usages  ⤴ albertoarriola
def output_student_and_course_names(student_data: list):
    """ This function displays the student and course names to the user

    :param student_data: list of Student rows to be displayed

    :return: None

    ChangeLog: (Who, When, What)
    Alberto Arriola, 3/9/2025, Created function
    """
    if len(student_data) == 0:      # if statement triggered if student_data is
        print("There is no student data to display.")  # print message if there
    else:      # else statement triggered if there is data in student_data
        print("-" * 50)
        for student in student_data:
            print(f'Student {student.first_name} '
                  f'{student.last_name} is enrolled in {student.course_name}')
        print("-" * 50)

```

Fig. 1.20 The `output_student_and_course_names()` function outputs the student information.

Student information is passed to the function in the `student_data` list parameter. A nested if statement is used to check if there is any data in the list. If it is empty, the message “There is no student data to display.” is printed out and the code returns to the main body.

If the `student_data` List is not empty, then the else statement is triggered and a for Loop is used to print out information for all the students in the list.

The final function in the IO class is the `input_student_data()` function. The description of the function appears below the title.

```

@staticmethod 1 usage  albertoarriola
def input_student_data(student_data: list):
    """
    ...

    # try-except block for exception handling
    student = Student()    # declare local variable of type Student
    # while loop runs until user enters a valid first name
    while (True):
        student.first_name = input("Enter the student's first name: ") # call to Student constructor t
        if not student.first_name:    # check if student.first_name is empty
            continue    # continue in while loop if student.first_name is empty
        else:
            break    # break out of loop if student.first_name has a value
    # while loop runs until user enters a valid last name
    while (True):
        student.last_name = input("Enter the student's last name: ")    # call to Student constructor
        if not student.last_name:    # check if student.last_name is empty
            continue    # continue in while loop if student.last_name is empty
        else:
            break    # break out of loop if student.last_name has a value
    student.course_name = input("Please enter the name of the course: ")    # call to Student construc
    student_data.append(student)    # append student to student_data list
    print()
    print(f"You have registered {student.first_name} {student.last_name} for {student.course_name}.")
    return student_data    # return student_data list to main body

```

Fig. 1.21 The `input_student_data()` function handles the user's input of student information.

The variable `student` of type `Student` is declared locally at the beginning of the function. The `input()` function asks the user to enter the student's first name and assigns it to the `first_name` attribute of the student `Student` object. The `first_name` setter method in the `Person` class uses `isalpha()` to determine if the name contains anything other than alpha letters. If it does, then a message is displayed saying, "The student's first name should only contain letters." If this happens, then the value is not assigned to `student.first_name`. Back in the `input_student_data` function, a nested while Loop is used to determine if `student.first_name` is empty. If it is empty, then the while Loop is continued and the user is asked to enter the student's first name again. If a valid first name is entered and `student.first_name` contains a value, then the else statement is triggered and the code breaks out of the while Loop.

Similar code is used for the student's last name. Again, the user will be continually asked to enter a last name until a valid one is assigned to `student.last_name`.

And finally, an `input()` statement is used to get the `student.course_name`. After the student first name, last name and course name are successfully assigned to the student `Student` object, it is appended to the `student_data` List. The student information is displayed and the `student_data` List is returned to the main body of code.

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` and the `students` List. The results read from the external JSON file and returned by the `read_data_from_file()` function are assigned to the `students` List variable.

```
# Start of main body

# Call the read_data_from_file() function from the FileProcessor class
# Send file_name str and student_data list as parameters; returned value is assigned to s
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
```

Fig. 1.22 Call to `read_data_from_file()` function in the `FileProcessor` class.

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 `output_menu()` function in the `IO` class is called and the `MENU` constant is passed as a parameter. This prints out the menu. Then the `input_menu_choice()` function in the `IO` class is called to record the user choice. The result is returned and assigned to the `menu_choice` String variable.

```
# Present and Process the data
while (True):

    # Call to output_menu() function in the IO class
    # Send menu str as parameter
    IO.output_menu(menu=MENU)

    # Call to input_menu_choice() function in the IO class
    # returned value is assigned to menu_choice str
    menu_choice = IO.input_menu_choice()
```

Fig. 1.23 Start of while Loop printing out the menu and asking for user input.

Then an if-elif-else statement is used to perform the appropriate action based on the user’s input.

```

# If-elif-else statements to process user's menu selection
# Register a student for a course
if menu_choice == "1": # If statement triggered if user enters "1"
    # Call to input_student_data() function in IO class
    students = IO.input_student_data(student_data=students)
    continue # return to start of while loop

# Show current data
elif menu_choice == "2": # elif statement triggered if user enters "2"
    # Call to output_student_and_course_names() function in IO class
    IO.output_student_and_course_names(students)
    continue # return to start of while loop

# Save data to a file
elif menu_choice == "3": # elif statement triggered if user enters "3"
    # Call to write_data_to_file() function in FileProcessor class
    FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
    continue # return to start of while loop

# Exit the program
elif menu_choice == "4": # else statement triggered if user enters "4"
    break # break out of the while loop

print("Program Ended") # Print out message that the program has ended

```

Fig. 1.24 if-elif block processes the user's menu selection.

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 functions 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_and_course_names()` 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”, then the `break` statement is used to exit the while Loop, and the program ends. The message, “Program Ended” is displayed.

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.

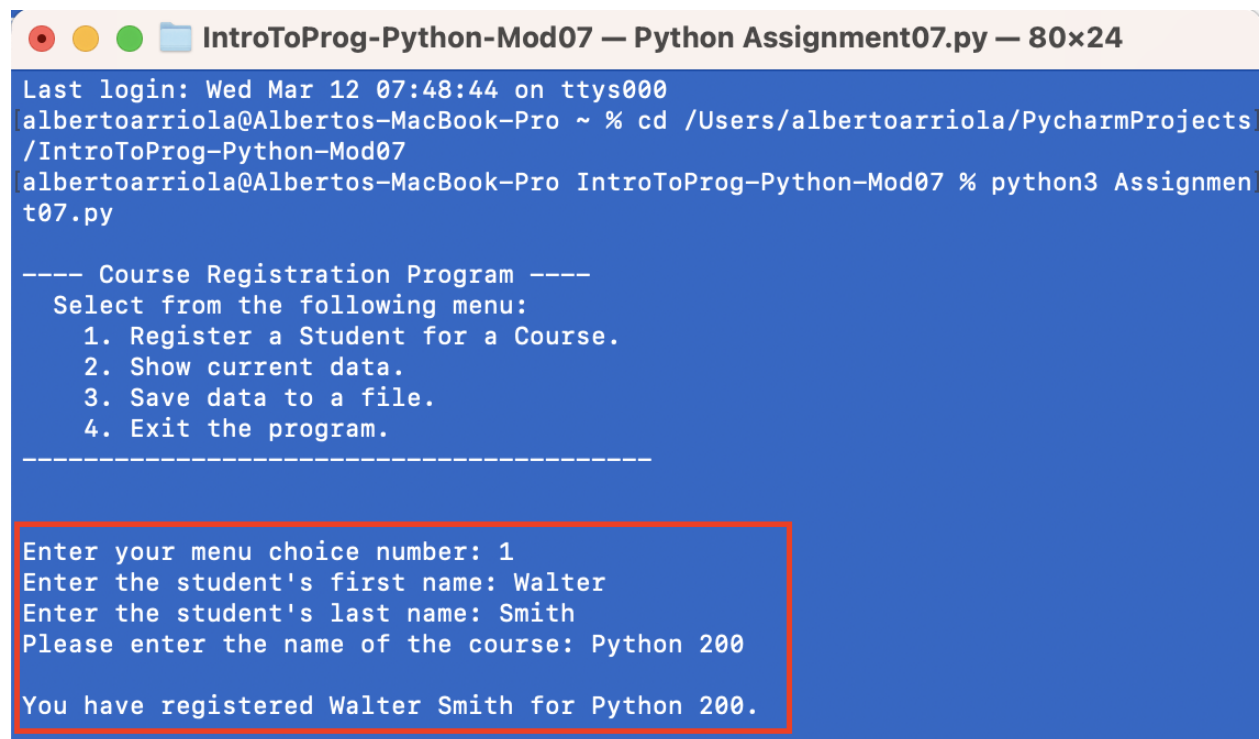
Testing the Program

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 Assignment07.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 "Walter". 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, a message was displayed saying that Walter Smith was registered for Python 200. Then I was returned to the Menu.



```
Last login: Wed Mar 12 07:48:44 on ttys000
albertoarriola@Albertos-MacBook-Pro ~ % cd /Users/albertoarriola/PycharmProjects/IntroToProg-Python-Mod07
albertoarriola@Albertos-MacBook-Pro IntroToProg-Python-Mod07 % python3 Assignment07.py

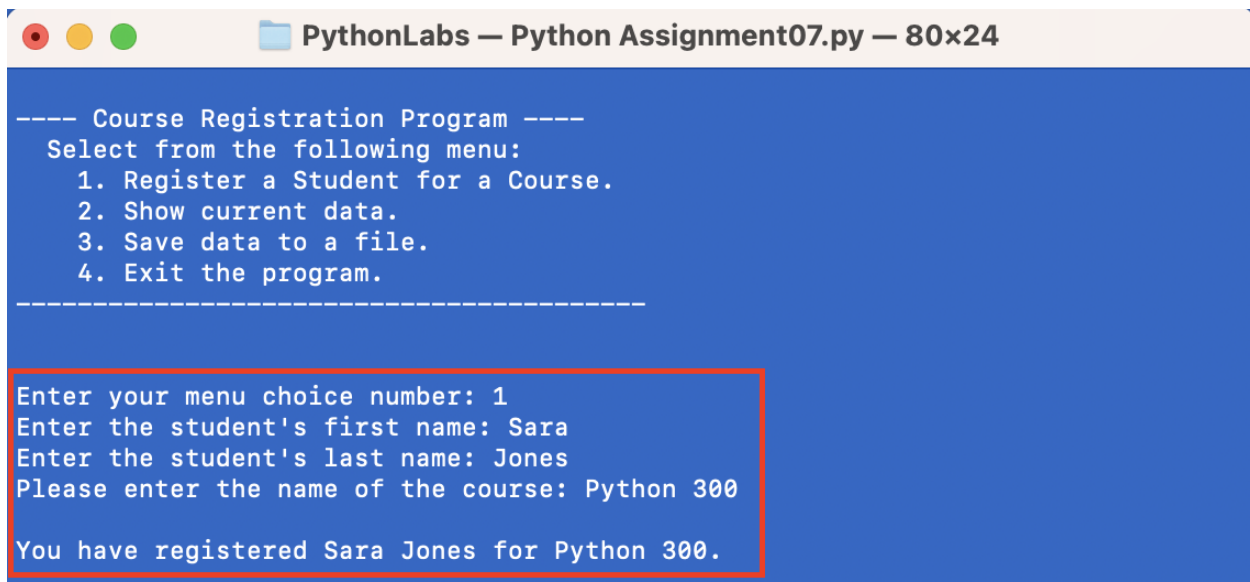
---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----

Enter your menu choice number: 1
Enter the student's first name: Walter
Enter the student's last name: Smith
Please enter the name of the course: Python 200

You have registered Walter Smith for Python 200.
```

Fig. 1.25 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". A message was displayed saying that Sara Jones was registered for Python 300. Again, I was returned to the Menu.



```
PythonLabs — Python Assignment07.py — 80x24

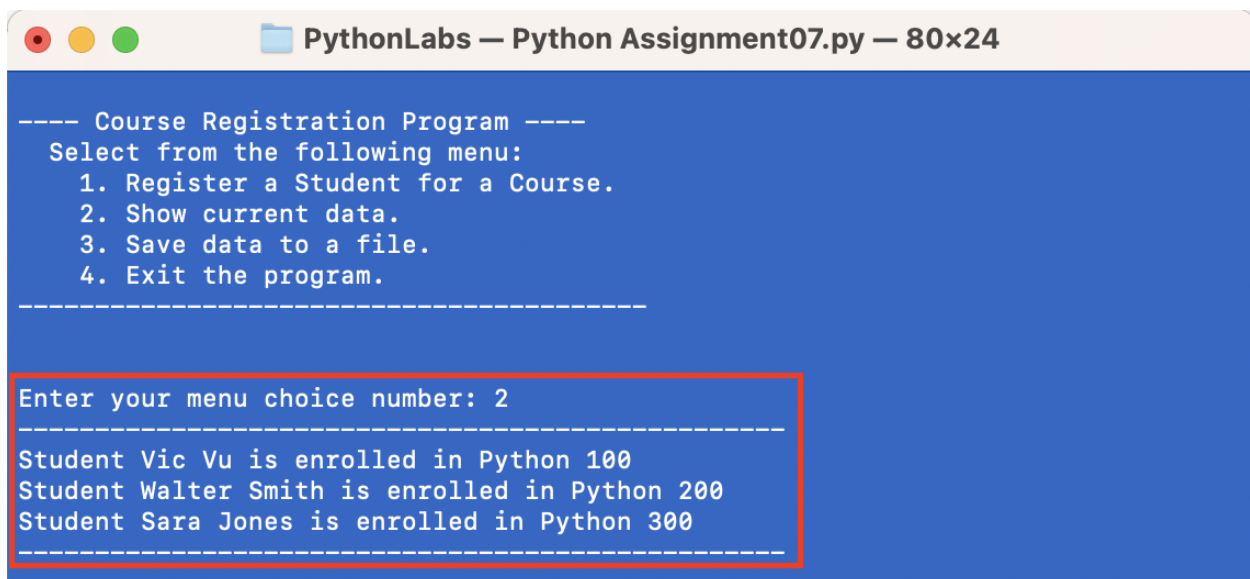
---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----

Enter your menu choice number: 1
Enter the student's first name: Sara
Enter the student's last name: Jones
Please enter the name of the course: Python 300

You have registered Sara Jones for Python 300.
```

Fig. 1.26 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 student that was 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 Assignment07.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.
-----

Enter your menu choice number: 2
-----
Student Vic Vu is enrolled in Python 100
Student Walter Smith is enrolled in Python 200
Student Sara Jones is enrolled in Python 300
-----
```

Fig. 1.27 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.
-----

Enter your menu choice number: 3
-----

Student Vic Vu is enrolled in Python 100
Student Walter Smith is enrolled in Python 200
Student Sara Jones is enrolled in Python 300
-----
```

Fig. 1.28 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.

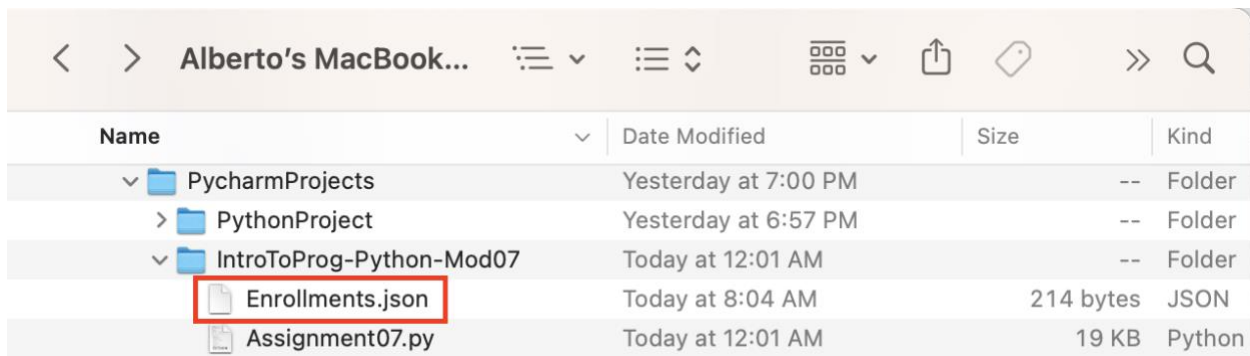


Fig. 1.29 Enrollments.json file in IntroToProg-Python-Mod07 folder.

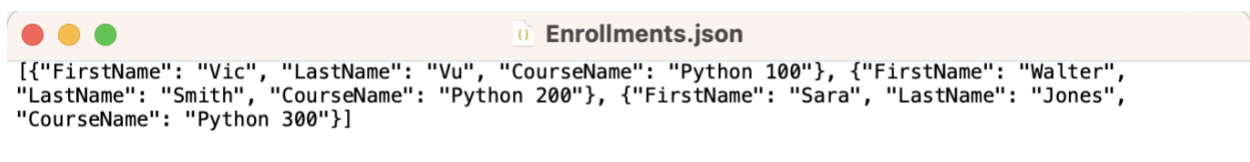


Fig. 1.30 Data in the Enrollments.json file.

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.
-----

Enter your menu choice number: 4
Program Ended
albertoarriola@Albertos-MacBook-Pro IntroToProg-Python-Mod07 %
```

Fig. 1.31 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.

```
IntroToProg-Python-Mod07 — Python Assignment07.py — 80x24
Last login: Wed Mar 12 11:06:00 on ttys000
albertoarriola@Albertos-MacBook-Pro ~ % cd /Users/albertoarriola/PycharmProjects/IntroToProg-Python-Mod07
albertoarriola@Albertos-MacBook-Pro IntroToProg-Python-Mod07 % python3 Assignment07.py
Error: There was a problem with reading the file.

-- 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.
-----

Enter your menu choice number: 
```

Fig. 1.32 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.  
-----  
  
Enter your menu choice number: 1  
Enter the student's first name: Vi3  
The student's first name should only contain letters.  
  
Enter the student's first name: Vi*  
The student's first name should only contain letters.  
  
Enter the student's first name: 
```

Fig. 1.33 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: Vic  
Enter the student's last name: Vu7  
The student's last name should only contain letters.  
  
Enter the student's last name: Vu!  
The student's last name should only contain letters.  
  
Enter the student's last name: 
```

Fig. 1.34 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.

```
IntroToProg-Python-Mod07 — Python Assignment07.py — 80x24

Enter your menu choice number: 5
Please, choose only 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.
-----

Enter your menu choice number: ?
Please, choose only 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.
```

Fig. 1.35 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 “Program ended” and ended.

PyCharm test

I also ran the script in PyCharm and verified that each of the selections behaved as expected.

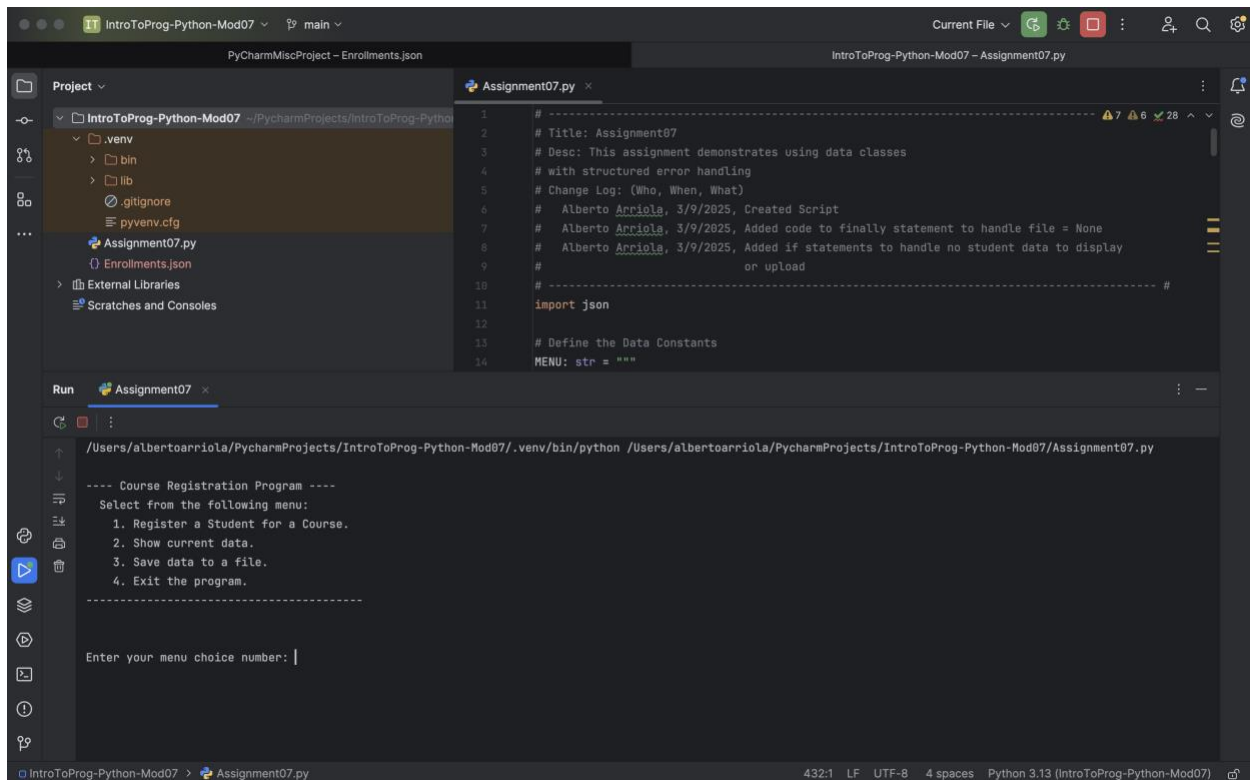


Fig. 1.36 Execution of the script in PyCharm.

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. 1.37 The uploaded data is written to the Enrollments.json file.

GitHub

I created a new project in PyCharm called “IntroToProg-Python-Mod07” and added it to my GitHub account. Then I shared the project on GitHub as a public repository. I added my assignment for Module 07, committed it to GitHub and verified that it appeared correctly. Here is the link to the repository: <https://github.com/albertoarriola/IntroToProg-Python-Mod07>.

Summary

Module 7 was about using functions and classes to encapsulate specific processes in the code. Creating a Person class and a Student sub-class allowed me to work with creating a custom object class and using inheritance to share certain attributes.

The classes were separated into data, processing and presentation sections according to the concept of Separation of Concerns. By using functions to encapsulate reusable sections of code in these classes, the main body of code became much easier to read and follow.

I also learned how to share my project on GitHub and created a new repository for it. I practiced committing and pushing my local changes to the project in GitHub and verified that I could view the History of my changes. If I decided to collaborate with other developers on my assignment, I could create a branch separate from the main project, work on it, and merge my changes without worrying about overwriting another developer's changes.