

Allan Duldulao

June 02, 2025

Foundations of Programming: Python

Assignment 07

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

Classes and Objects

Introduction

Now that we have learned about functions, parameters, arguments, organizing code properly, DocStrings and Separations of Concern on the last module, we are now ready to tackle Classes and Objects. In this module, as we work with objects and classes, we will also learn on working with Constructors, Data validation, code inheritance and GIT. We will use our learning in this module and also pass modules to achieve the objective of Assignment 07.

My Script for the Assignment

Started the assignment with the given starter file. And work slowly editing the script line-by-line, until achieved the objective of the assignment. First, I took my module 06 assignment and by reading the module and doing the lab, and also downloading the starter code I edit my old code to look like the lab codes for this module. I started first with creating a Class.

Class

A class is a method to for creating objects. It defines a set of attributes and methods that the objects created from the class will have. Classes are fundamental part of OOP in python. Key features of a class are Attributes, Methods, Encapsulation, Inheritance and Polymorphism. First objective of this assignment is creating a Person class. **Figure 1**, we created class for the assignment. On **Figure 2**, we also created Student class that inherits Person class.

```
# TODO Create a Person Class
class Person: 1 usage
    """
    A base class representing student first name and last name.
    ChangLog: (Who, When, What)
    ADuIdulao: 5/30/2025, Created person class
    """
```

Fig. 1: class Person created

```
# TODO Create a Student class the inherits from the Person class
class Student(Person):
    """
    A class representing student data, inheriting from Person.
```

Fig. 2: class Student created

Constructors

To continue with the assignment, after creating a class we need to work on constructor. A constructor is a special method that is automatically called when an object of a class is created. Its primary purpose is to set an object's attributes values when it's created. Constructors are sometimes called an initializer because it's set up necessary initial data for the object. **Fig. 3**, we created a constructor class that takes student first name and student last name as parameters with a default value of empty strings and assigning them to **self.student_first_name** and **self.student_last_name**.

```
# TODO Add first_name and last_name properties to the constructor
def __init__(self, student_first_name: str = '', student_last_name: str = ''):
    """
    Initialize a Person object with first and last name
    ChangeLog: ADuldulao, 5/31/2025, Created Method docstrings
    """
    self.student_first_name = student_first_name
    self.student_last_name = student_last_name
```

Fig. 3: Added first name and last name properties to the constructor

__init__, is the constructor. It allows to define and initialize the attributes of object when it's created. We also use self to access or modify the attributes of the instance. Here we assigns instance attribute for the student first and last name. self, ensures the methods and attributes are tied to the correct object.

Properties

Properties are functions designs to manage attribute data. Typically, each attribute, you create two properties, one for getting the data and setting the data. These functions are commonly known as “Getter” and “Setter” or formally as “Accessors” and “Mutators”. Getter property, functions enable you to access the data while optionally applying formatting. Setter property, functions allow you to add validation and error handling. It is important to remember that the setter decorator and function name must match the name of the getter for them to work. **Figure 4**, we created setter and getter for student first name and student last name and over ride the **__str__()** method to return to Person data. If you noticed the **2**

underscores before the attributes name, this marks the attribute as private and indicates that developers should not access the attribute from code outside of the class. When an attribute is private, it can not be changed by code outside of the class.

```
# TODO Create a getter and setter for the first_name property
@property
def student_first_name(self):
    return self.__student_first_name.title()

@student_first_name.setter
def student_first_name(self, value: str):
    if value.replace(_old: " ", _new: "").isalpha() or value == "":
        self.__student_first_name = value
    else:
        raise ValueError("The first name should not contain numbers.")

# TODO Create a getter and setter for the last_name property
@property
def student_last_name(self):
    return self.__student_last_name.title()

@student_last_name.setter
def student_last_name(self, value: str):
    if value.replace(_old: " ", _new: "").isalpha() or value == "":
        self.__student_last_name = value
    else:
        raise ValueError("The first last should not contain numbers.")
# TODO Override the __str__() method to return Person data
def __str__(self):
    return f'{self.student_first_name},{self.student_last_name}'
```

Fig. 4: Created a getter and setter for first name and last name

As have shown in **Fig. 2**, we created a class Student that inherits from the Person class. On **Fig. 5**, we call the Person constructor and pass the first name and the last name data. We use `super()` to access parent (Person) class method. This is especially helpful when overriding methods in the child class but still needing to use parents' class. By using `super()`, we can write cleaner, more modular, and maintainable code. We also added "Getter" and "Setter" for course name. We also have to over ride the `__str__()` method to return the Student data.

```

# TODO call to the Person constructor and pass it the first_name and last_name data
def __init__(self, student_first_name: str = '', student_last_name: str = '', course_name: str = ''):
    super().__init__(student_first_name=student_first_name, student_last_name=student_last_name)
# TODO add a assignment to the course_name property using the course_name parameter
    self.__course_name = course_name

# TODO add the getter for course_name
@property 15 usages (12 dynamic)
def course_name(self):
    return self.__course_name.title()

# TODO add the setter for course_name
@course_name.setter 12 usages (12 dynamic)
def course_name(self, value: str):
    self.__course_name = value

# TODO Override the __str__() method to return the Student data
def __str__(self):
    return f'{self.student_first_name},{self.student_last_name},{self.course_name}'

```

Fig. 5: Call Person constructor

Next step is creating class FileProcessor. This is for writing and reading data to/from a JSON file. This class provides static method to read student data from JSON file into a list of Student objects, and to write a list of Student objects to a JSON file as a list of dictionaries. This class FileProcessor was made for the assignment on the module 06. But in this part of the class, since we slowly modifying assignment 06 to achieve the objective of module 07, we have 2 TODO for module 7 assignment on class FileProcessor. **Fig. 6**, is to add code to convert dictionary data to student data and **Fig. 7**, to add code to convert Student objects to dictionaries.

```

# TODO replace this line of code to convert dictionary data to Student data
for student_dict in list_of_dictionary_data:
    student_object: Student = Student(student_first_name=student_dict["FirstName"],
                                       student_last_name=student_dict["LastName"],
                                       course_name=student_dict["CourseName"])
    students.append(student_object)
file.close()

```

Fig. 6: Code to convert dictionary data to student data

```

# TODO Add code to convert Student objects into dictionaries (Done)
list_of_dictionary_data: list = []
for student in students:
    student_json: dict \
    = {"FirstName": student.student_first_name, "LastName": student.student_last_name,
       "CourseName": student.course_name}
    list_of_dictionary_data.append(student_json)

```

Fig. 7: Code to convert Student objects into dictionaries

As we have stated this script was from the last module, so we also have the same class IO, where we have a collection of layer functions that manage user input and output. Here we defined several functions such as `output_error_messages`, `output_menu`, `input_menu_choice`, `output_student_courses`, and `input_student_data`, and uses `@staticmethod` decorator. In this part of the script of the objective, we have to satisfy 2 TODO or objective for the assignment. First, **Fig. 8**, in the function `input student data`, we need to replace the previous code to use a `Student` objects instead a dictionary objects. And on **Fig. 9**, added a code to access `Student` object data instead a dictionary.

```
# TODO Replace this code to use a Student objects instead of a dictionary objects
student = Student(first_name, last_name, course_name)
print(f"You have registered {student.student_first_name} {student.student_last_name}"
      f" for {student.course_name}.")
students.append(student)
```

Fig. 8: Replaced code to use Student objects instead of dictionary

```
for student in students:
    # TODO Add code to access Student object data instead of dictionary data
    message = "{}, {}, {}".format(*args: student.student_first_name,
                                   student.student_last_name,
                                   student.course_name))
    print(message.format(*args: student.student_first_name,
                         student.student_last_name,
                         student.course_name))

print("-" * 47)
print()
```

Fig. 9: Add code to access Student object data

And after getting all the TODO objectives satisfied, **Fig. 10**, is the rest of the script where we present and process the data using while loop.

```
while (True):
    IO.output_menu(menu=MENU)
    menu_choice = IO.input_menu_choice()
    # Present the menu of choices # Input user data
    if menu_choice == "1": # This will not work if it is an integer!
        students = IO.input_student_data(student_data=students)
        continue
    # Present the current data
    elif menu_choice == "2":
        IO.output_student_courses(student_data=students)
        # Process the data to create and display a custom message
        continue
    # Save the data to a file
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue
    # Stop the loop
    elif menu_choice == "4":
        break # out of the loop
    else:
        print("Please select only 1,2, and 3. Select 4 to exit")
        continue
print("Program Ended")
```

Fig. 10: Present and Process Data

Testing and Results

After writing the script and making it work. Now it's time to see the different results to meet the assignment objective. First, **Fig. 11**, we run our script on the command prompt.

```
C:\Users\allan\Documents\Python\PythonCourse\_Module07\Assignment>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.
-----

What would you like to do:
1
What is the student's first name?
Sue
What is the student's last name?
Jones
Please enter the name of the course:
Python 100
You have registered Sue Jones for Python 100.

---- 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.
-----

What would you like to do:
2
The current registration data:
-----
Vic,Vu,Python 100
Sue,Jones,Python 100
-----

---- 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.
-----

What would you like to do:
3
The following student was added to list!
Student Sue Jones is enrolled in Python 100

---- 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.
-----

What would you like to do:
4
Program Ended

C:\Users\allan\Documents\Python\PythonCourse\_Module07\Assignment>pause
Press any key to continue . . . |
```

Fig. 11: Script ran in command prompt

On this test, the program takes the user's input of student's first, last and course name, displays it and saved it. And it also shows allowing multiple registration. On **Fig. 12**, the program saves the user's input of first name, last name and course name to a JSON file.


```
Enrollments.json x Assignment07-Starter.py Mod07Lab03.py
1  [
2    {
3      "FirstName": "Vic",
4      "LastName": "Vu",
5      "CourseName": "Python 100"
6    },
7    {
8      "FirstName": "Sue",
9      "LastName": "Jones",
10     "CourseName": "Python 100"
11   }
12 ]
```

Fig. 12: User's data saves in JSON file

```
C:\Users\allan\Documents\Python\PythonCourse\_Module07\Assignment>Assignment07.py
Text file must exist before running this script!

-----Technical Error Message-----
[Errno 2] No such file or directory: 'Enronts.json'
File not found.
<class 'FileNotFoundError'>
To prevent confusion. PRESS 4 TO EXIT IMMEDIATELY!!!

---- 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.
-----

What would you like to do:
```

Fig. 13: Error handling when there is no file to read

On Figure 13, I purposely edit the file name to test the error handling and it worked as written. I put some additional message to warn users of possible complication if the wish to continue. On the next figure, Figure 15, we will test the error handling capability of the script when the users enter a non-alphabet character on the first name and last name inputs. During testing of this code, I find a problem when the user has 2 names, so edited the script to accept 2 names for the first name. Figure 14, added code to accept spaces for the name.

```
@student_first_name.setter
def student_first_name(self, value: str):
    if value.replace(_old: " ", _new: "").isalpha() or value == "":
        self.__student_first_name = value
    else:
        raise ValueError("The first name should not contain numbers.")

create a getter and setter for the last name property
```

Fig. 14: Making sure code will only fail on non-alphabet

```
C:\Users\allan\Documents\Python\PythonCourse\_Module07\Assignment>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.  
-----
```

```
What would you like to do:  
1  
What is the student's first name?  
Se7en  
That value is not the correct type of data
```

```
-----Technical Error Message-----  
The first name should not contain numbers.  
Inappropriate argument value (of correct type).  
<class 'ValueError'>
```

```
---- 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.  
-----
```

```
What would you like to do:  
1  
What is the student's first name?  
Paul Jake  
What is the student's last name?  
Elleven  
That value is not the correct type of data
```

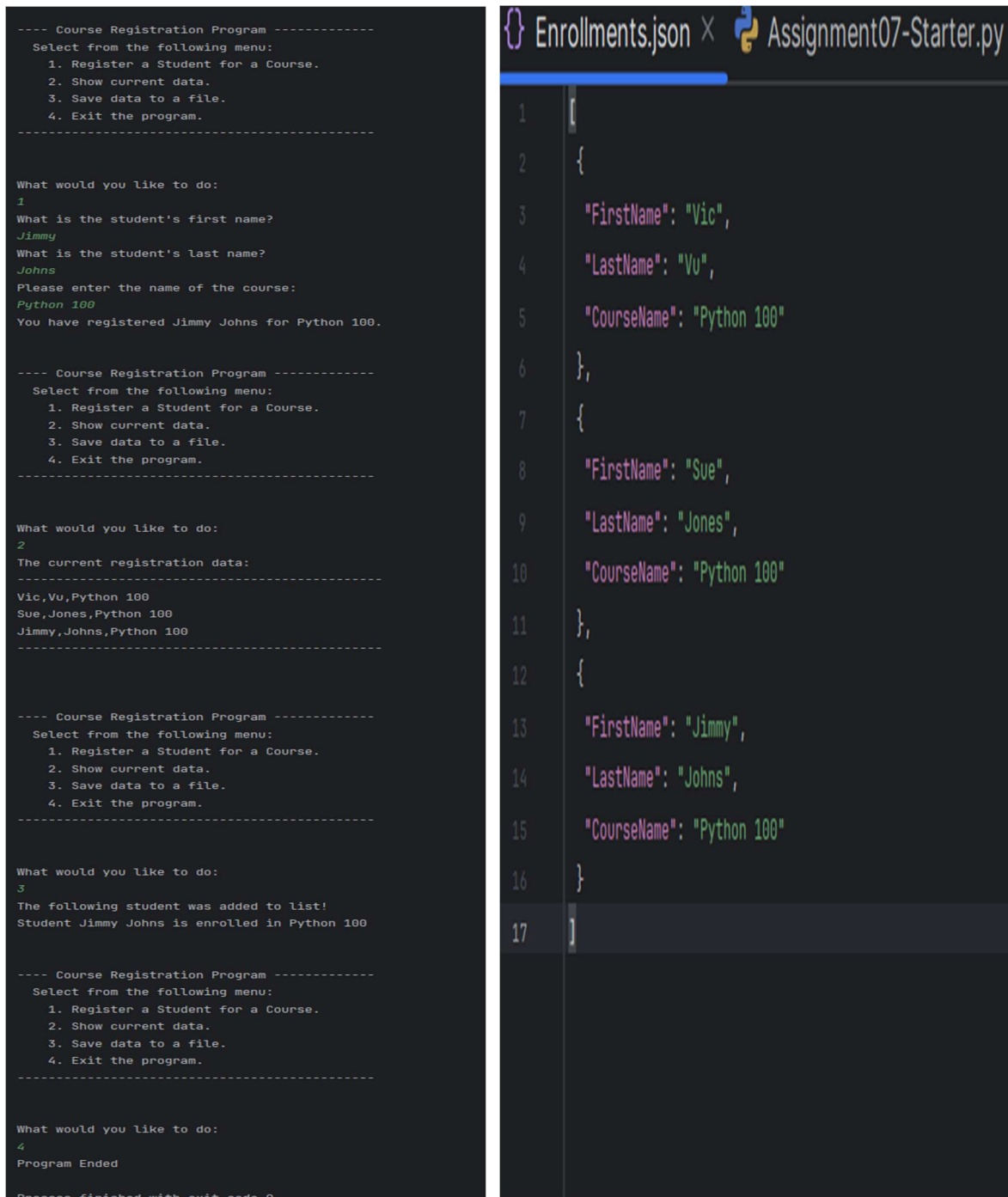
```
-----Technical Error Message-----  
The last name should not contain numbers.  
Inappropriate argument value (of correct type).  
<class 'ValueError'>
```

```
---- 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.  
-----
```

```
What would you like to do:  
1  
What is the student's first name?
```

Fig. 15: error handling when non-alphabet is entered by user

After testing our code on the command prompt. We will test now the code in PyCharm.



The image shows a PyCharm IDE window with two panes. The left pane displays the output of a Python script, and the right pane displays the content of a file named 'Enrollments.json'.

Left Pane (Terminal Output):

```
---- 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.
-----

What would you like to do:
1
What is the student's first name?
Jimmy
What is the student's last name?
Johns
Please enter the name of the course:
Python 100
You have registered Jimmy Johns for Python 100.

---- 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.
-----

What would you like to do:
2
The current registration data:
-----
Vic,Vu,Python 100
Sue,Jones,Python 100
Jimmy,Johns,Python 100
-----

---- 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.
-----

What would you like to do:
3
The following student was added to list!
Student Jimmy Johns is enrolled in Python 100

---- 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.
-----

What would you like to do:
4
Program Ended

Process finished with exit code 0
```

Right Pane (Enrollments.json):

```
[
  {
    "FirstName": "Vic",
    "LastName": "Vu",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Sue",
    "LastName": "Jones",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Jimmy",
    "LastName": "Johns",
    "CourseName": "Python 100"
  }
]
```

Fig. 16: Results in PyCharm

On the menu choice 3, I purposely write my script to show just the last student information to avoid too many names being displayed. I accomplished this accessing the index of the last name on list by using `[-1]`.

Summary

After reading, watching and attending the zoom meeting module 07, we have finally better understanding on Classes and Objects, and also our knowledge and skills in Python is getting better and broader. Although the assignments are getting harder and harder especially for me who has no knowledge coming in, writing this assignment script makes me more understand how the functions, classes, objects, constructors and other works.