

Sashi Chintala

December 3<sup>rd</sup>, 2024

Course name: IT FDN 110 A Au 24: Foundations Of Programming: Python

Assignment 07

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

# Classes and objects

## Introduction

In this assignment 07, I will share how I was able to write the program using the topics covered from the notes on the basic components like statements, functions, and classes. Learned that any program uses three components which are statements, functions, and classes as many as developer creates.

Understood the differences between objects and classes and magic method of python. Learned these topics from the notes and just wanted to point out some main ones.

**Constructor:** From the notes, a constructor is a special method (function) 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 is created. Constructors are sometime called an initializer because it sets up any necessary initial data (known as state) for the object.

**Abstraction** is the concept of simplifying complex reality by creating classes based on the essential properties and behaviors its objects should have. Abstraction focuses on defining what an object does rather than how it does it.

**Encapsulation** is a common way to perform abstraction. It is the act of hiding the internal details and implementation of a class by bundling the data (attributes or properties) and the methods (functions) that operate on that data into a single unit called a class. It restricts direct access to some of an object's components and prevents the accidental modification of data.

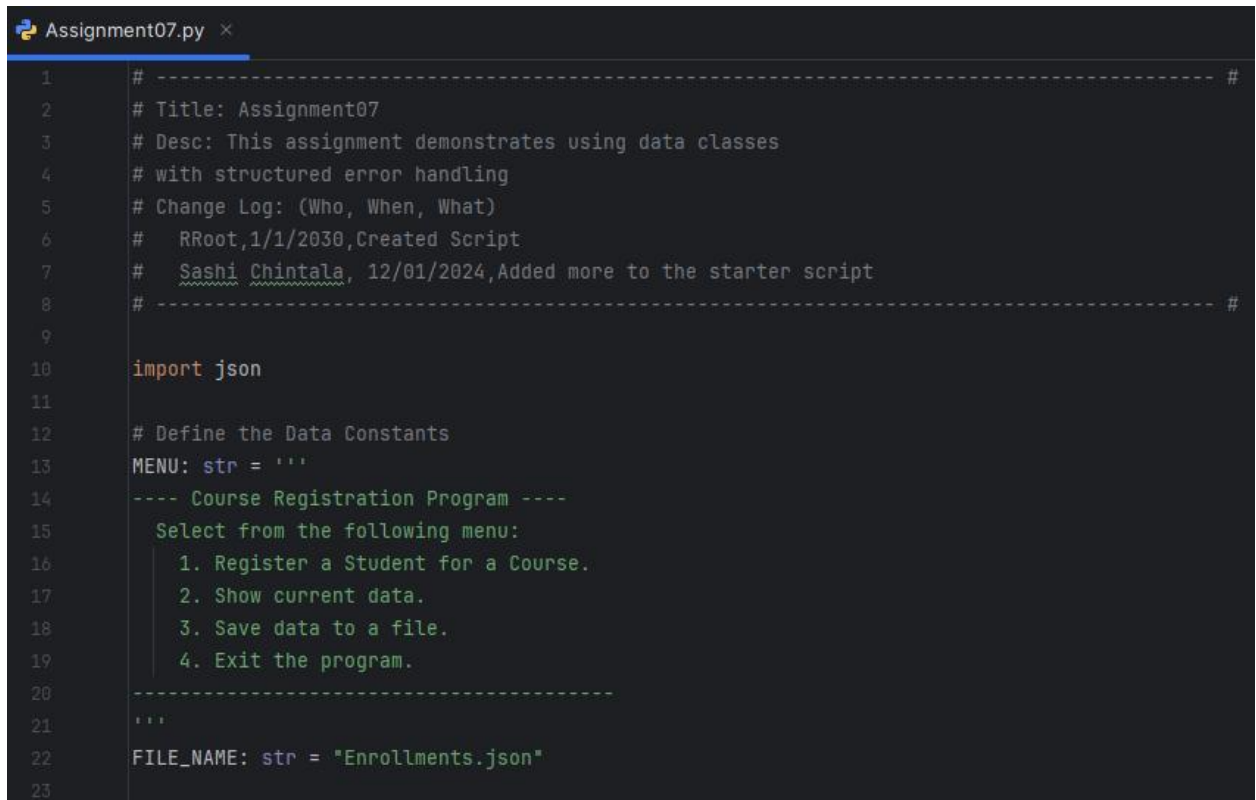
**Inheritance** is a core concept in OOP where a new class (child class) can inherit data and behaviors (properties and methods) from an existing class (parent class). "Inherited code" refers to code that is inherited or derived from another source, such as a parent class or codebase from a previously developed project. Inherited code serves as the foundation for building new code or extending existing functionality. In our program we are using the inherited code from person class.

## Creating the program

Thanks to Professor Root as he gave us the starter script from the previous module 06 assignment and I started building it. In our module03, 04, and 05 programs, we took student information and used it for student registration. Information regarding interactive console using the prompt and using the four menu choices and we will add more error checking, file handling, duplicate student check, course validation, and display a message at the end showing registrations.

## Constants

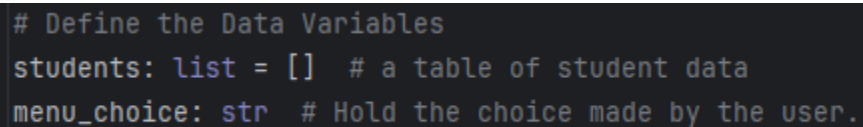
**Figure 1.1:** Constants



```
1 # ----- #
2 # Title: Assignment07
3 # Desc: This assignment demonstrates using data classes
4 # with structured error handling
5 # Change Log: (Who, When, What)
6 #   RRoot,1/1/2030, Created Script
7 #   Sashi Chintala, 12/01/2024, Added more to the starter script
8 # ----- #
9
10 import json
11
12 # Define the Data Constants
13 MENU: str = ''
14 ---- Course Registration Program ----
15     Select from the following menu:
16         1. Register a Student for a Course.
17         2. Show current data.
18         3. Save data to a file.
19         4. Exit the program.
20 -----
21 '''
22 FILE_NAME: str = "Enrollments.json"
23
```

## Variables

**Figure 1.2:** list of variables used



```
# Define the Data Variables
students: list = [] # a table of student data
menu_choice: str # Hold the choice made by the user.
```

Students list to store student data, and menu\_choice to store user's menu choice 1-4.

We will create a Person class which is called a base class for any user with a first name and last name. Later in the program we will create Student class which inherits the properties from the Person class and adds the information about the course name.

```

33     # TODO Override the __str__() method to return Person data (Done)
34
35     @↓ class Person: 1 usage
36     @↓     def __init__(self, first_name:str, last_name:str):
37         self.first_name = first_name # cannot use outside person class
38         self.last_name = last_name
39
40         @property 2 usages
41         def first_name(self)->str:
42             """
43             Returns the first_name as a title
44             :return: The first_name, formatted as a title
45             """
46             return self._first_name.title()
47
48         @first_name.setter 1 usage
49         def first_name(self, value:str)->None:
50             """
51             Sets the first name, while doing the validations
52             :param value: The value to set
53             """
54             if value.isalpha():
55                 self._first_name = value
56             else:
57                 raise ValueError("First name must be alphabetic")
58
59         @property 2 usages
60         def last_name(self) -> str:
61             """
62             Returns the last_name as a title
63             :return: The last name, formatted as a title
64             """
65             return self._last_name.title()
66
67         @last_name.setter 1 usage
68         def last_name(self, value: str) -> None:
69             """
70             Sets the first name, while doing the validations
71             :param value: The value to set
72             """

```

```

    """
    if value.isalpha():
        self._last_name = value
    else:
        raise ValueError("Last name must be alphabetic")

def __str__(self) -> str: #__str__ is called magic method for python. print will call this method instead
    """
    The string function for person
    :return: The string as a csv value
    """
    return f"{self.first_name},{self.last_name}"

class Student(Person): #inherit everything from person class 9usages
    def __init__(self, first_name:str, last_name:str, course_name:str):
        super().__init__(first_name,last_name)
        self._course_name=course_name

    @property 4 usages
    def course_name(self) -> str:
        """
        returns the course_name
        :return: course_name
        """
        return self._course_name

    @course_name.setter
    def course_name(self, value) -> None:
        self._course_name = value

    def __str__(self) -> str:
        """
        the string function for a person
        :return: the string as a csv value
        """
        return f'{super().__str__()}, {self.course_name}'

```

**Figure 1.3-4** shows the Person and Student class and setter

Constructor (---init---) defines the variables and information it stores.

The @Property is nothing but a decorator that tells about the getter method to fetch the value and setter method to validate the content.

The program utilizes two classes:

### Person Class

This is the base class representing individuals. It contains:

- Attributes: first\_name: The first name of the person, last\_name: The last name of the person.
- Methods: Getters and setters for first\_name and last\_name with validation to ensure names are alphabetic.

- `__str__` method for string representation of a person's name in CSV format.

## Student Class

This class inherits from Person and extends it to include:

- Additional Attribute: `course_name`: The name of the course the student is enrolled in.
- Overridden `__str__` Method: Extends the parent class's method to include the course name.

## FileProcessor Class

In FileProcessor Class, added two functions, `read_data_from_file`: Reads data from a JSON file and loads it into a list, and `write_data_to_file`: Writes data from a list to a JSON file. Benefits of Using a FileProcessor Class are code reusability, error handling, and maintainability.

**Figure 1.5** Shows the class *FileProcessor*

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

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    """

    @staticmethod 1 usage
    def read_data_from_file(file_name: str, student_data: list[Student])->list[Student]:
        """ This function reads data from a json file and loads it into a list of dictionary rows

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function

        :param file_name: string data with name of file to read from
        :param student_data: list of dictionary rows to be filled with file data

        :return: list
        """
        file_data = []
        file=None
        try:
            file = open(file_name, "r")
            file_data = json.load(file)
            file.close()
        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)

        finally:
            if file is not None and not file.closed:
                file.close()
        for row in file_data:
            student_data.append(
                Student(row['first_name'],row['last_name'],row['course_name'])
            )
        return student_data
```

## IO Class

The IO class manages user interaction. Its features include:

- **Menu Display:** Presents the user with a menu of choices.
- **Input Validation:** Ensures valid menu selections and data entries.
- **Error Messages:** Provides clear and detailed error messages when issues arise.
- **Data Display:** Outputs all registered students and their courses in a formatted manner.

Figures 1.6-7 Class *IO*

```
# Presentation ----- #
class IO: 10 usages
    """
    A collection of presentation layer functions that manage user input and output

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    RRoot,1.2.2030,Added menu output and input functions
    RRoot,1.3.2030,Added a function to display the data
    RRoot,1.4.2030,Added a function to display custom error messages
    """

    @staticmethod 5 usages
    def output_error_messages(message: str, error: Exception = None):
        """ This function displays the a custom error messages to the user

        ChangeLog: (Who, When, What)
        RRoot,1.3.2030,Created function

        :param message: string with message data to display
        :param error: Exception object with technical message to display

        :return: None
        """
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message -- ")
            print(error, error.__doc__, type(error), sep='\n')
```

```

class IO: 10 usages

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

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function

        :return: string with the users choice
        """
        choice = "0"
        try:
            choice = input("Enter your menu choice number: ")
            if choice not in ("1","2","3","4"): # Note these are strings
                raise Exception("Please, choose only 1, 2, 3, or 4")
        except Exception as e:
            IO.output_error_messages(e.__str__()) # Not passing e to avoid the technical message

        return choice

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

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function

        :param student_data: list of dictionary rows to be displayed

        :return: None
        """

        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)

```

## Error handling

In this module we learned how to handle the errors using the try-except blocks. Please see Figure 1.3, we will create a new file in the enrollments.json doesn't exist and to move further we continue the program by not stopping it. Also, we will validate the student inputs to be alphabetic and will display a message requesting user to enter only alphabets using the `str.isalpha()`.

```

# Present and Process the data
while (True):

    # Present the menu of choices
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    # 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_and_course_names(students)
        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 only choose option 1, 2, or 3")

print("Program Ended")

```

**Figure1.8** error handling

The program also shows how to collect input from students and double checks for errors before writing to the file. We will ensure only data is accepted for students by first name and last name as alphabetical and not numeric. Used a while true loop and it executes the program repeatedly till the condition is met with the menu choices. We use input prompts enabling users to enter the information from menu choice 1-4 till we use option 4 to break the program.

1. On menu choice 1, the program prompts the user to enter the student's first name and last name, followed by the course name, using the input () function and stores the inputs in the variables (student's first name and last name, followed by the course name). The print function will display the student registration details.

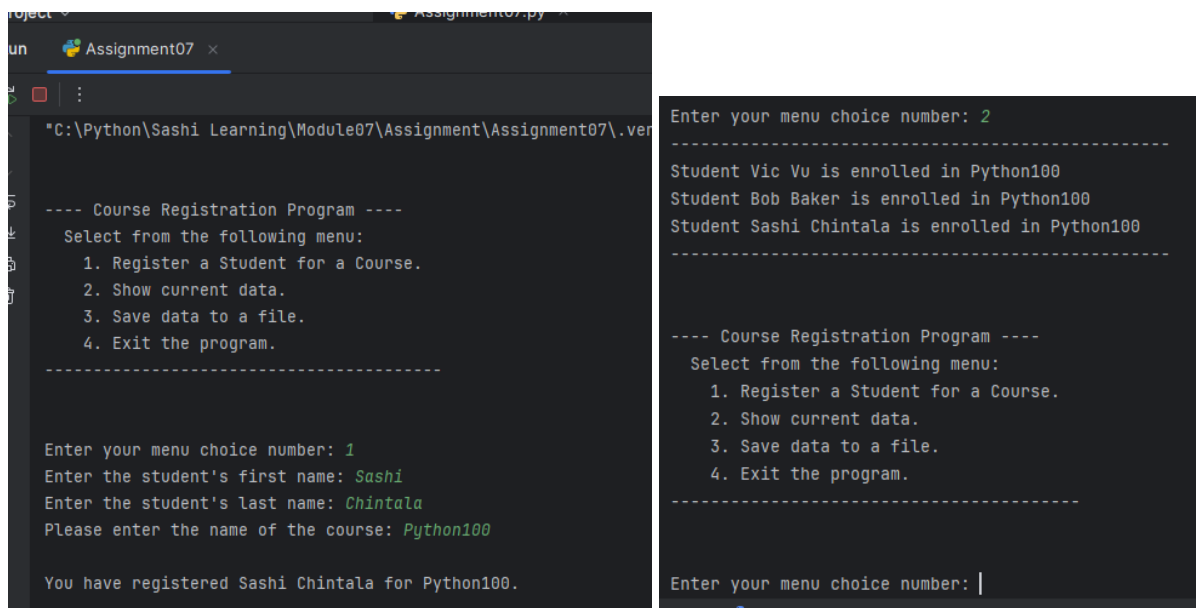


2. On menu choice 2, the presents a coma-separated string by formatting the collected data using the print () function. We are using a loop for getting student data from students multidimensional list for first name, last name, and course name, and print the data.
3. On menu choice 3, program opens the file named "Enrollments.json" in write mode using the open () function. It writes the content of the json\_data variable to the file using the write() function, then file is closed using the close() method. Then displays what was stored in the file. Using \n we will write the data in the new line instead of deleting the existing. Similar to the above step, we will use for loop for write the student details to the file for multiple students.
4. On the menu, 4, program prints the message that we are ending the program and stops the while loop and ends the program.

## Testing and results from PyCharm

*Results from PyCharm IDE for menu choice 1-4 from the two screenshots, and json output.*

**Figures 2.1 -2.3** – menu choice 1 -4, registers students, displays, writes, and exits



```
*C:\Python\Sashi Learning\Module07\Assignment\Assignment07\venv
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: Sashi
Enter the student's last name: Chintala
Please enter the name of the course: Python100

You have registered Sashi Chintala for Python100.

Enter your menu choice number: 2
-----
Student Vic Vu is enrolled in Python100
Student Bob Baker is enrolled in Python100
Student Sashi Chintala is enrolled in Python100
-----

---- 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: |
```

```
---- 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 Python100
Student Bob Baker is enrolled in Python100
Student Sashi Chintala is enrolled in Python100
-----

---- 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

Process finished with exit code 0
```

**Figure 2.4** shows the enrollments.json file data

```
Assignment07.py  Enrollments.json x
1 [{"first_name": "Vic", "last_name": "Vu", "course_name": "Python100"},
2 [{"first_name": "Bob", "last_name": "Baker", "course_name": "Python100"},
3 [{"first_name": "Sashi", "last_name": "Chintala", "course_name": "Python100"}]]
```

## Testing the program from Command prompt on Windows

I was able to test the same from command prompt.

### Summary:

As usual I am grateful for the notes and videos shared by Professor Root. Learned OOP (Object-Oriented Programming) basics and differences between statements, functions, and classes. The importance of constructor and how the class inheritance works. I am still learning most of the topics from the notes and doing the labs again. Compared to previous modules, this module took longer than usual to study and understand the topics.