Jiahua (Joey) Zhang

6/4/2025

IT FDN 110 A

Assignment 07

GitHub Link

Classes and Objects with Knowledge Check

Introduction

This module expands on classes and objects. It starts with highlighting the key differences between statements, functions and classes and digs into data, presentation and processing classes. Module 7 also teaches constructors, attributes and properties which are cruxes to object-oriented programming that promotes flexibility. There is a Python script exercise below that highlights all the concepts that I learned reading module 7.

Statements, Functions, and Classes

Statements are any written instructions in the code and are organized into functions when they need to be used many times within a program.

Similarly, functions are organized into classes when they need to be used multiple times in the program. This is an example of modularity to keep the code organized and easy to read and maintain.

Objects vs. Classes

The last module taught how to use functions within a class using the @staticmethod decorator but an object instance can also be created from a class if one wants to access unique object data. This is called a custom class. See example (Mode07-Notes, Page 4):

```
class Person:
    first_name: str = ''
    last_name: str = ''



person1 = Person()
person1.first_name = "Vic"
person1.last_name = "Vu"

person2 = Person()
person2.first_name = "Sue"
person2.last_name = "Jones"
```

```
print(person1.first_name, person1.last_name, person1)
print(person2.first_name, person2.last_name, person2)
```

In this case, each object created from a class maintains its own set of data and has its own memory also known as data encapsulation. This keeps each object independent (**isolation**), turns the class as a template (**reusability**), and promotes **abstraction** which is interacting with objects based on their interfaces without worrying about the backend.

Data Classes vs. Processing Classes and Data Class Components

The three buckets that a class can fall under are data, processing or presentation (IO) classes. Processing and presentation classes usually only have methods which is another term for function, but data classes can also include **attributes, constructors and properties**.

An **attribute** is a variable that holds specific object data and can have different data types. Each class object can have its own set of attributes where the attributes store the characteristics of the object.

A **constructor** is a special method that is automatically called when an object of a class is created. The primary purpose is to set or initialize an object's attributes when it is created. They are automatically called when the class object is created. In Python, the constructor is named "__init__" and never have a return type.

Properties are functions designed to manage attribute data by using the "get" data and "set" data property functions. The @property decorator is used as the "getter" function and @name_of_property.setter decorator is used as the "setter" function.

The **"self"** keyword is used to refer to data or functions in an object instance. When the class loads into memory, it can have many object instances of the class or copies of the class code. To identify which copy is referenced, **"self"** is used in this way to signify separate uniqueness.

Adding Data Validation and Private Attributes

A private attribute cannot be changed by code outside of the class and uses two underscores before the attribute's name. Data validation can be done on this through the setter's constructor properties. See example below (Mod07-Notes, Page 16):

```
class Person:

    def __init__(self, first_name: str = "", last_name: str = ""):  #
parameters default to empty
        self.first_name = first_name  # set the attribute using the property
to provide validation
        self.last_name = last_name  # set the attribute using the property to
provide validation

    @property  # (Use this decorator for the getter or accessor)
    def first_name(self):
        return self.__first_name.title()  # Optional formatting code

    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha() or value == "":  # allow characters or the default
empty string
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")
```

Inherited Code

Inherited code is code that his derived from another source or parent where the parent class passes down its attributes to the child class. It serves as the foundation for building new code or building off existing functionality. In Python, the top-level class is typically named “object” which is not to be confused for objects that are derived from the object class.

This is otherwise known as Python’s Magic Methods where Python’s “object” (parent) class have two notable built-in methods: __int__() constructor and __str__() method. These double underscore methods are automatically invoked by the Python interpreter in response to specific operations. The __str__() method is responsible for returning readable string data of an object. This helps simplify common tasks and allows customization.

The override method is used to change the way an inherited method works. To do add new properties to a class, the properties are added to the initializer constructor’s parameters and set as attributes.

Working with Custom Objects and Converting Back to JSON

When a custom object is created, it will contain structured data as class instances vs. unstructured dictionaries in a dictionary object. Using a list of dictionary objects in a JSON file, we can convert this dictionary list into a list of custom objects and vice versa. The JSON module cannot handle custom objects so that’s why converting goes both ways to use custom objects in a JSON file.

Converting JSON dictionary objects to custom objects (Mod07-Notes, Page 29):

```
# Using this starting data
student_first_name = 'Vic'
student_last_name = 'Vu'
gpa = 3.8

# We have learned how to create a list of dictionary objects
json_students = []
student = {"FirstName": student_first_name,
           "LastName": student_last_name,
           "GPA": gpa}
json_students.append(student)
```

```
# We can convert the list of dictionaries into a list of Student objects like this
student_objects = []
for row in json_students:
    student_obj = Student(first_name=row["FirstName"],
                          last_name=row["LastName"],
                          gpa=row["GPA"])
    student_objects.append(student_obj)
```

Converting custom objects to JSON (Mod07-Notes, Page 30):

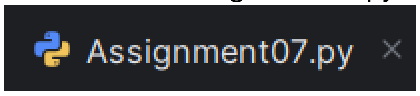
```
json_students = []
for student in student_objects:
    student_dict = {"FirstName": student.first_name,
                    "LastName": student.last_name,
                    "GPA": student.gpa}
    json_students.append(student_dict)

file = open("test.json", "w")
json.dump(json_students, file)
file.close()
```

Git vs Github

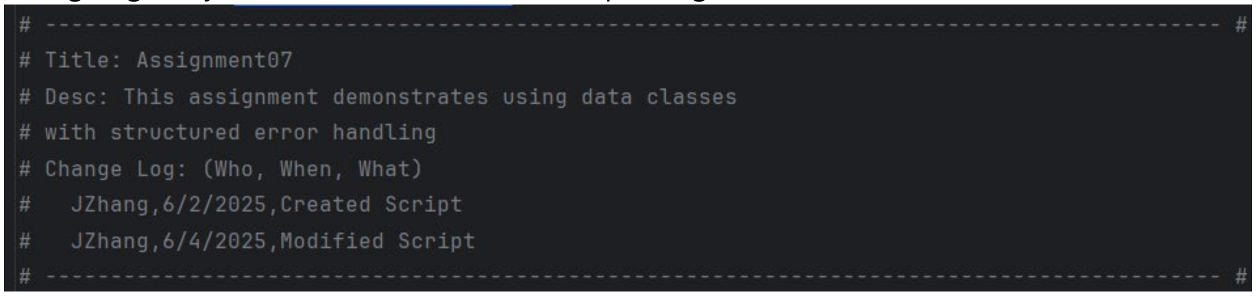
Git is a software that manages version control of one or more files. It allows file cloning and changes to the clone while maintaining the original copy. Github is a web-based platform that works in conjunction with Git.

Assignment07.py Knowledge Check

File Name
File is named Assignment04.py to signify that this is a python program.


Assignment07.py

Script Header
For proper documentation, I have included the following header to display the title, description, and change log to my name as the creator of the script along with the date.


```
# ------------------------------------------------------------------------------- #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
#   JZhang,6/2/2025,Created Script
#   JZhang,6/4/2025,Modified Script
# ------------------------------------------------------------------------------- #
```

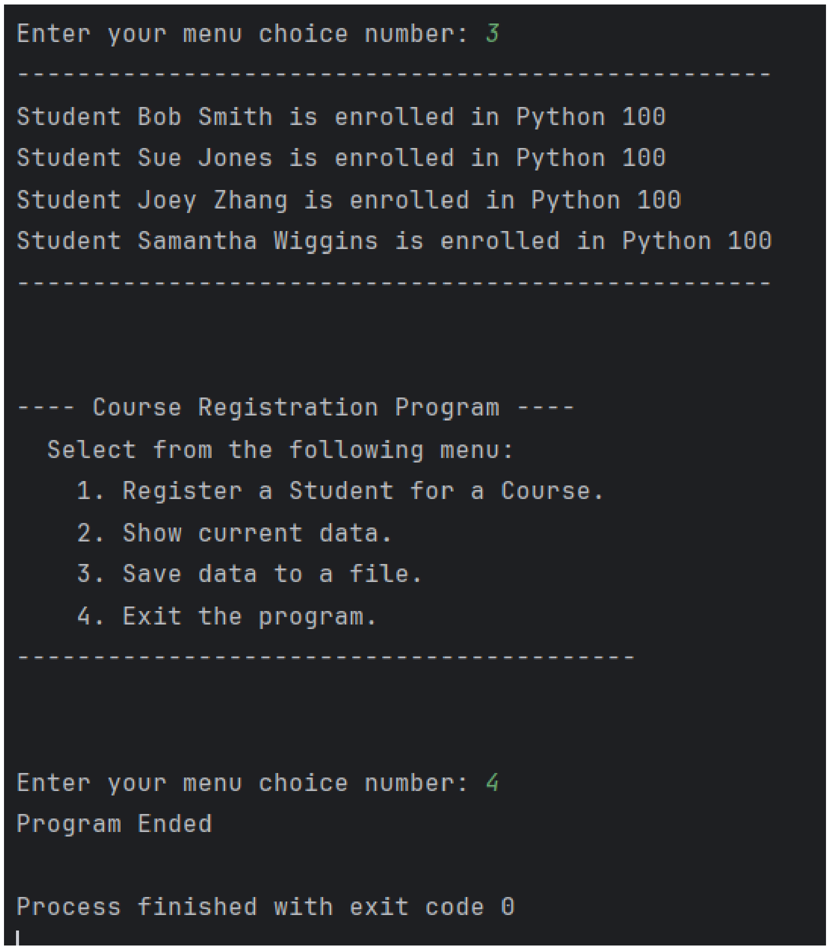
Body

Following the instructions, I created two custom classes (Person class (object class) and Student class (child class)) to build on assignment 06. I added class attributes first_name and last_name to the Person and Student classes and included the course_name attribute to the Student class as an override specific to it.

All of the above properties also include validation code to ensure quality control and error-handling. The program also includes JSON dictionary list conversion to student objects in the read_data_from_file function and student class conversion back to JSON dictionary objects in the write_data_from_file function. The code is also updated to call directly from the student class instead of dictionary data. The __str__ function is used to extract comma separated data from each data class.

The last step is to print the file data so the user can see the displayed data and the loop ends stopping the program.

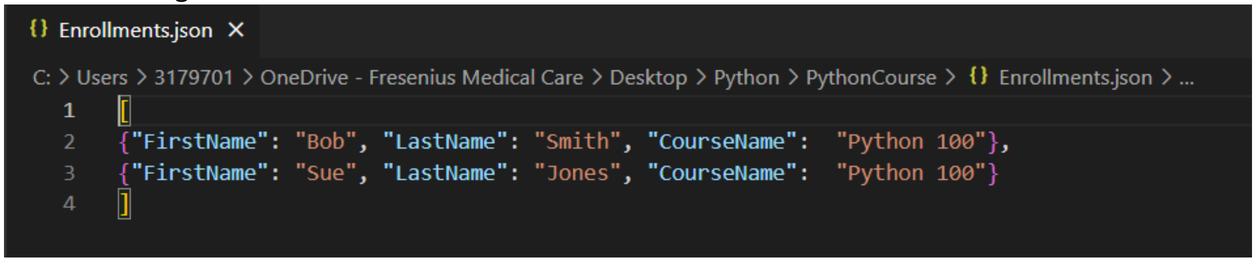
Test
Confirm that the program takes and displays the user's inputs and saves it on top of the existing data in Enrollments.json where multiple registrations are allowed.



```
Enter your menu choice number: 3
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Joey Zhang is enrolled in Python 100
Student Samantha Wiggins 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.
-----------------------------------------



Enter your menu choice number: 4
Program Ended

Process finished with exit code 0
```

PyCharm Test with JSON output

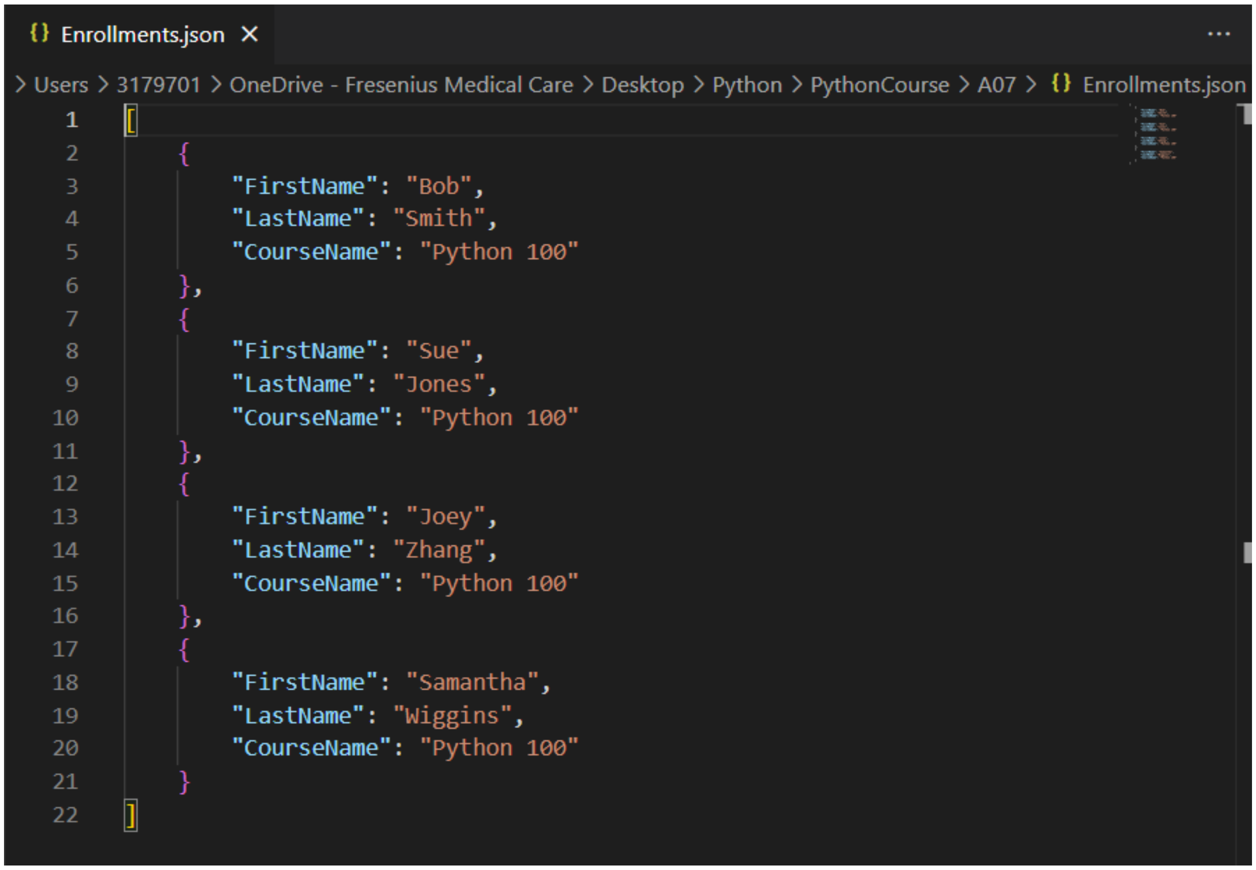
Before running code:



{} Enrollments.json ×
C: > Users > 3179701 > OneDrive - Fresenius Medical Care > Desktop > Python > PythonCourse > {} Enrollments.json > ...

```
[
{"FirstName": "Bob", "LastName": "Smith", "CourseName":  "Python 100"},
{"FirstName": "Sue", "LastName": "Jones", "CourseName":  "Python 100"}
]
```

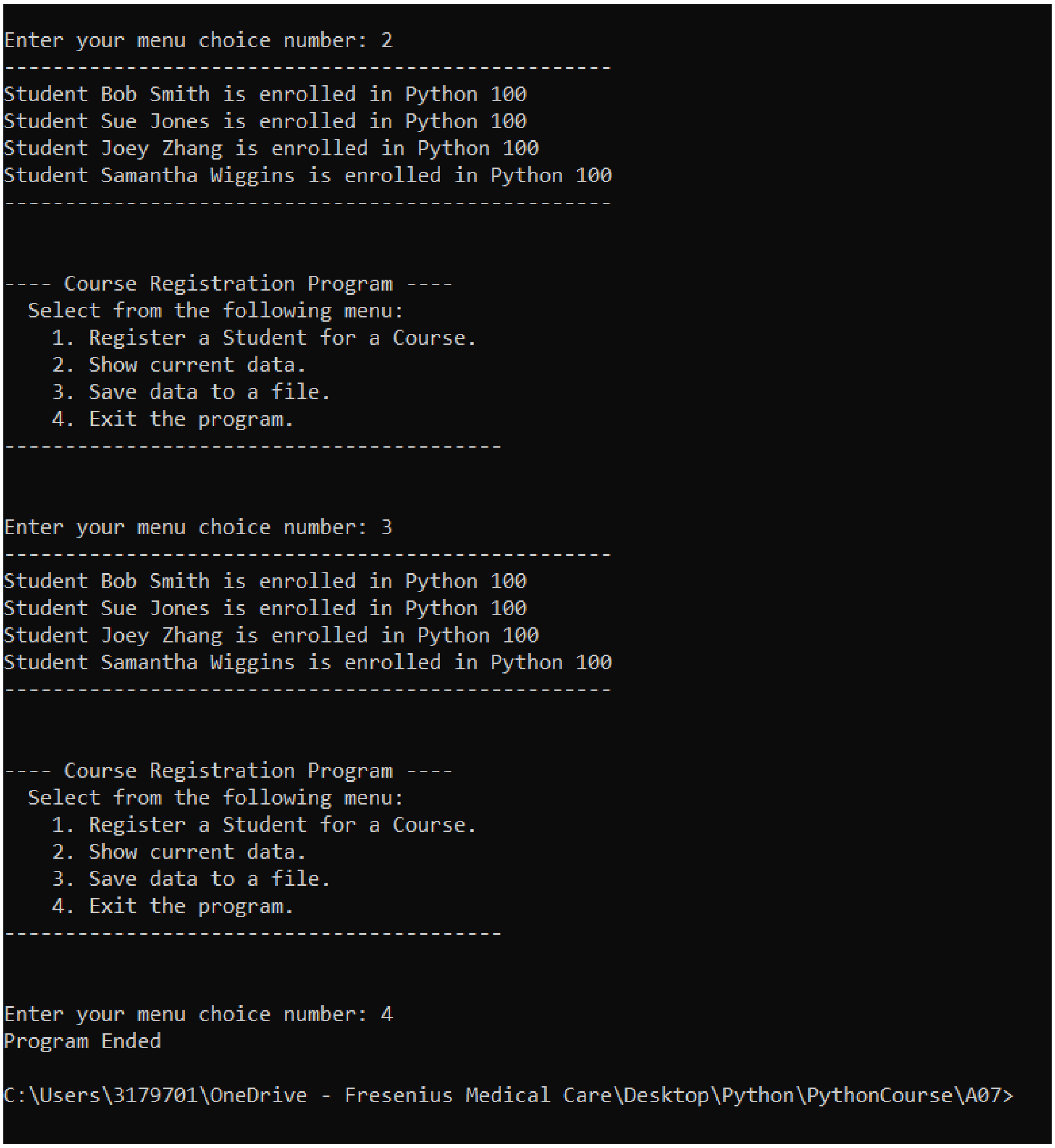
After running code:



{} Enrollments.json ×
> Users > 3179701 > OneDrive - Fresenius Medical Care > Desktop > Python > PythonCourse > A07 > {} Enrollments.json

```
[
    {
        "FirstName": "Bob",
        "LastName": "Smith",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "Sue",
        "LastName": "Jones",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "Joey",
        "LastName": "Zhang",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "Samantha",
        "LastName": "Wiggins",
        "CourseName": "Python 100"
    }
]
```

Command Terminal Test (see next page):



```
Enter your menu choice number: 2
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Joey Zhang is enrolled in Python 100
Student Samantha Wiggins 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.
-----------------------------------------



Enter your menu choice number: 3
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Joey Zhang is enrolled in Python 100
Student Samantha Wiggins 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.
-----------------------------------------



Enter your menu choice number: 4
Program Ended

C:\Users\3179701\OneDrive - Fresenius Medical Care\Desktop\Python\PythonCourse\A07>
```

Summary

The seventh module of this Introduction to Python course focuses on custom object classes and how it is used as a template to access unique data. Concepts such as attributes, constructors (getter and setter methods) and properties are highlighted to illustrate the key differences between data classes and process and presentation classes. The module also points out private attributes which are used to encapsulate proper data validation and error handling. Finally, the parent-child inheritance concept is introduced where the parent class (object) pass down attributes to the child classes.