Yuying Xie
June 4, 2025
Class: IT FDN 110 A
Assignment 07

Classes and Objects

Introduction

This project demonstrates the use of object-oriented programming principles in Python, focusing on class creation, inheritance, encapsulation, data validation, and file handling. Through the development of Person and Student classes, the project shows how to organize and manage student registration data effectively using constructors, setters/getters, and structured error handling. It also includes reading and writing JSON files, converting between objects and dictionaries, and updating program logic to align with class-based design. This work was completed in Visual Studio Code and shared via GitHub.

Key Concept

1. **Statements** are the fundamental units of a Python program that perform actions like assigning values, modifying data, displaying output, and controlling flow. Types include data statements (data = 1), processing (data += 1), presentation (print(data)), and control flow (if, while, break). And control flow statements such as conditional statements (if, else) and looping statements (while, break, continue), etc[1].
2. **Functions** are reusable code blocks that group related statements to perform specific tasks. Defined with def, they improve organization, reduce repetition, and enhance readability. Functions can take parameters, return results, and be called multiple times with different inputs[2].
3. **Classes** are templates or blueprints used to group together related functions (called methods) and data (called attributes)[3].
4. **Objects** are instances of classes and represent specific realizations of the template defined by a class. Each object has its own copy of the class's attributes, allowing you to work with multiple distinct sets of data[4].
5. **Encapsulation** refers to the concept that each object maintains its own set of data in memory, independent from other objects[5].
6. **Attributes** are variables that belong to an object and store information about its state. For example, a Person class might have first_name and last_name attributes to represent a person's name. Each object created from the class can have its own unique values for these attributes[6].

---

[1] Mod07-Notes, P2
[2] Mod07-Notes, P3
[3] Mod07-Notes, P4
[4] Mod07-Notes, P4-6
[5] Mod07-Notes, P17
[6] Mod07-Notes, P6; P14

7. **Static methods** are special methods within a class that do not depend on the object's attributes. They are defined using the @staticmethod decorator and can be called directly on the class without creating an object instance[7].
8. **A constructor** is a special method automatically called when an object is created. Its main job is to initialize the object's attributes with defined values (state)[8].
9. **Data validation** can be managed by adding checks inside the constructor and raising errors when data is invalid[9].

Creating the Person Class

To begin the project, I created the Person class (see figure1) to manage basic identity information. This class includes a constructor that initializes first_name and last_name as private attributes. I implemented getter and setter methods for each attribute, using validation logic in the setters to ensure only alphabetic characters or empty strings are allowed. I also noticed that in Person.__init__ if I use self.__first_name = first_name directly. That bypasses the setter. Instead, I should call the setter in __init__: self.first_name = first_name to invokes setter.

```
27  # Create a Person Class
28  class Person:
29      """
30      A collection for personal information process
31
32      ChangeLog: (Who, When, What)
33      Yuying Xie, 6/2/2025, Created class
34      """
35  # Add first_name and last_name properties to the constructor
36      def __init__(self, first_name:str = "", last_name:str = ""):
37          self.first_name = first_name
38          self.last_name = last_name
39  # Create a getter and setter for the first_name property
40      @property
41      def first_name(self):
42          return self.__first_name.title()
43      @first_name.setter
44      def first_name(self, value):
45          if value.isalpha() or value == "":  # is character or empty string
46              self.__first_name = value
47          else:
48              raise ValueError("The first name should not contain numbers.")
49  # Create a getter and setter for the last_name property
50      @property
51      def last_name(self):
52          return self.__last_name.title()
53      @last_name.setter
54      def last_name(self, value):
55          if value.isalpha() or value == "":  # is character or empty string
56              self.__last_name = value
57          else:
58              raise ValueError("The last name should not contain numbers.")
59  # Override the __str__() method to return Person data
60      def __str__(self):
61          return f'{self.first_name} {self.last_name}'
```

Figure 1. Create person class

[7] Mod07-Notes, P9
[8] Mod07-Notes, P6
[9] Mod07-Notes, P13

Creating the Student Class with Inheritance

Next, I built the Student class (Figure 2), which inherits from the Person class. I used the super() function to call the parent class's constructor and initialize first_name and last_name. I then added a new attribute, course_name, along with corresponding getter and setter methods. The setter allows any string input and formats it to the title case in the getter. I overrode the __str__ method to return a formatted string showing which course the student is registered for. I also defined a to_comma_sep() method to produce a CSV-style string for each student's record.

```
63  # Create a Student class the inherits from the Person class
64  class Student(Person):
65      """
66      A collection for personal information process
67
68      ChangeLog: (Who, When, What)
69      Yuying Xie, 6/2/2025, Created class
70      """
71  # call to the Person constructor and pass it the first_name and last_name data
72      def __init__(self, first_name:str = "", last_name: str="", course_name:str = ""):
73          super().__init__(first_name, last_name)
74          self.__course_name = course_name
75  # add a assignment to the course_name property using the course_name parameter
76  # add the getter for course_name
77      @property
78      def course_name(self):
79          return self.__course_name.title()
80
81  # add the setter for course_name
82      @course_name.setter
83      def course_name(self, value):
84          self.__course_name = value
85  # Override the __str__() method to return the Student data
86      def __str__(self):
87          return f'{self.first_name} {self.last_name} is registered for {self.course_name}'
88
89      def to_comma_sep(self):
90          return f"{self.first_name},{self.last_name},{self.course_name}"
91
```

Figure 2. Create student class

Implementing File Reading and Object Conversion

In the FileProcessor class, I edited a static method read_data_from_file() that reads student data from a JSON file (Figure 3). It takes a list of Student objects and converts each one into a dictionary by accessing the first_name, last_name, and course_name attributes.

```python
class FileProcessor:
    """
    A collection of functions that read and write json file

    ChangeLog: (Who, When, What)
    Yuying Xie, 6/3/2025, Created class
    """
    @staticmethod
    def read_data_from_file(file_name: str):
        """ This function reads data from a json file and loads it into a list of dictionary rows
        then returns the list filled with student data.

        ChangeLog: (Who, When, What)
        Yuying Xie, 6/2/2025, Created class

        :param file_name: string data with name of file to read from

        :return: list - student_objects
        """

        try:
            # Get a list of dictionary rows from the data file
            file = open(file_name, "r")
            json_students = json.load(file)

            # Convert the list of dictionary rows into a list of Student objects
            student_objects = []
            # replace this line of code to convert dictionary data to Student data
            for student in json_students:
                student_obj = Student(
                    first_name = student["FirstName"],
                    last_name = student["LastName"],
                    course_name =student["CourseName"])
                student_objects.append(student_obj)

            file.close()
```

Figure 3. Implementing file reading and object conversion

Writing Objects Back to JSON Format

I also edited a write_data_to_file() method (Figure 4) in the FileProcessor class, which performs the reverse operation. It takes a list of Student objects and converts each one into a dictionary by accessing the first_name, last_name, and course_name attributes.

```python
@staticmethod
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

    ChangeLog: (Who, When, What)
    Yuying Xie, 6/3/2025, Created class

    :param file_name: string data with name of file to write to
    :param student_data: list of dictionary rows to be writen to the file

    :return: None
    """

    try:
        # Add code to convert Student objects into dictionaries
        list_of_dictionary_data:list = []
        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)

        file = open(file_name, "w")
        json.dump(list_of_dictionary_data, file)
        file.close()
        print("The following data was saved to file!")
        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:
        if file.closed == False:
            file.close()
```

Figure 4. Write objects back to JSON format

Updating User Interaction to Use Classes

Finally, I revised the IO class to make it compatible with the Person and Student classes. In input_student_data(), I replaced the use of plain dictionaries with actual Student object creation, allowing built-in validation in the setters to manage user input. The output_student_and_course_names() method was also updated to use each object's to_comma_sep() method to display data in a clean, comma-separated format. These updates brought the program into alignment with a structured, object-oriented design while ensuring smooth user interaction and data integrity.

Test Program

I used VScode as my IDE this time so I tested the program through VScode & Terminal. See result as appendix1 showed.

Post on GitHub

I created a new repository called "IntroToProg-Python-Mod07" (Figure 5), uploaded both of your files to the repository and committed the changes to save your work. The link is https://github.com/yuyingxieuw/IntroToProg-Python-Mod07.git

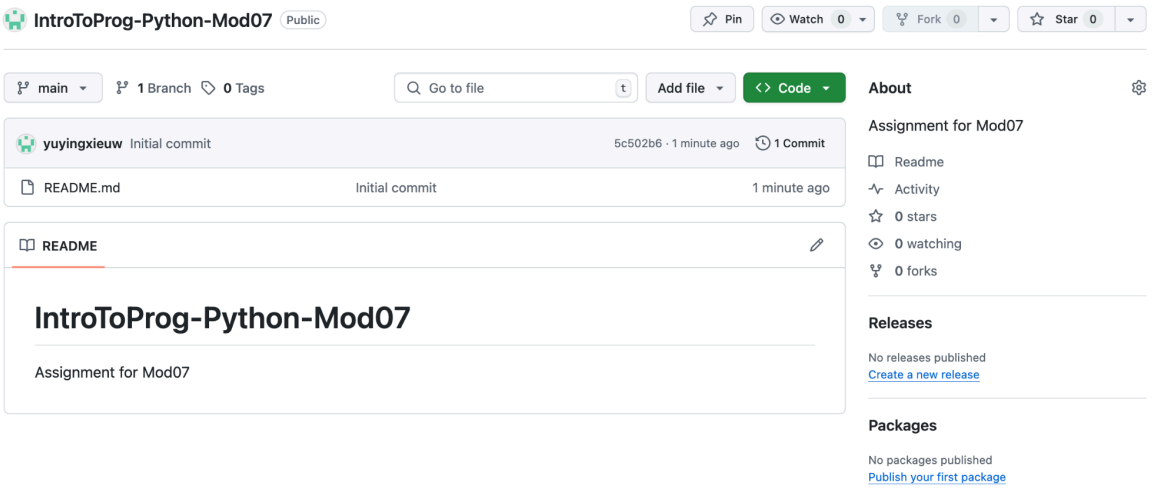

Figure 5. GitHub repository

Summary

In this assignment, I designed a Person class with encapsulated attributes and validation logic. I extended this by creating a Student class with an additional course attribute and a method to output data in comma-separated format. I modified the file processor to convert between JSON dictionaries and student objects, and updated the user interface layer to support object-based interaction.

Appendix 1.

```
---- 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
What is the student's first name? Testing
What is the student's last name? Testing
What is the name of the course? June4

Testing Testing is registered for June4


---- 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
------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
Testing,Testing,June4
------------------------------------------------


---- 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
The following data was saved to file!
------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
Testing,Testing,June4
------------------------------------------------


---- 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
What is the student's first name? Yuying
What is the student's last name? Xie
What is the name of the course? Python 101
```

Yuying Xie is registered for Python 101

---- 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

Bob,Smith,Python 100
Sue,Jones,Python 100
Testing,Testing,June4
Yuying,Xie,Python 101

---- 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
The following data was saved to file!

Bob,Smith,Python 100
Sue,Jones,Python 100
Testing,Testing,June4
Yuying,Xie,Python 101

---- 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
What is the student's first name? 4
What is the student's last name? 4
What is the name of the course? 5
One of the values was 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.
----------------------------------------


Enter your menu choice number: 2
------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
Testing,Testing,June4
Yuying,Xie,Python 101
------------------------------------------------


---- 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
The following data was saved to file!
------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
Testing,Testing,June4
Yuying,Xie,Python 101
------------------------------------------------
```

```
---- 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
.venv(base) xy@MacBook-Pro-683 Python Fundation101 %
```