Student Name - Roli Pathak

Class – IT FDN 110 : Foundations Of Programming: Python

Date – 3/12/2025

Assignment: Assignment 07

GitHub link - https://github.com/rolipathak/IntroToProg-Python-Mod07.git

Introduction: This module covers advanced topics in python Classes and functions, static method decorators. This module organizes programs code by utilizing object oriented programming, inheritance and encapsulation principles.

In Python, programs are built using three basic components: statements, functions, and classes. These components help organize and manage code in a way that makes it easier to develop and maintain by applying Object Oriented Programming concepts.

Statements

A statement is a line of code that tells the computer to do something. Common types of statements include data assignment, processing, presentation, looping, and conditional statements. Example – If then Elif…Else or while <condition> or For while loops. These help control how the program behaves and interacts with data.

```
while (True):

    IO.output_menu(menu=MENU)
    menu_choice = IO.input_menu_choice()
    if menu_choice == "1":  # This will not work if it is an integer!
        students = IO.input_student_data(student_data=students)
        continue
    elif menu_choice == "2":
        IO.output_student_and_course_names(students)
        continue
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue
    elif menu_choice == "4":
        break  # out of the loop
```

Functions

A function is a group of statements organized together to perform a specific task. Functions allow to reuse code, making the program cleaner and more organized. For example, we might group statements into a functions and call the function whenever needed, passing in different data each time.

```python
def output_student_and_course_names(student_data: list):
    """
        Shows list of all students and rgistered courses .
        Change Log:  RPathak,3/12/2025,Created Script
        """
    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)
```

Classes

A class is a blueprint for creating objects, which are instances of the class. A class groups related functions (called methods) and data (called attributes) into one structure. For example, we can create a Person class to represent people, where each person has attributes like first_name and last_name, and methods to perform actions. By organizing functions into classes, a program van be more manageable and reusable.

```python
class Person:  1 usage
    """
        Represents a generic Person with first name and last name.
        Change Log:  RPathak,3/12/2025,Created Script

        """
    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name

    @property
    def first_name(self):
        return self.__first_name.title()

    """Getter method to return the first name in title case."""

    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha():  # error handling if value is not alphabets
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")
    """Setter method to ensure the first name contains only alphabetic characters."""
```

Objects

An object is an instance of a class. It’s created using the class as a template, and each object can store its own unique data.

```
try:
    student_first_name = input("Enter the student's first name: ")
    if not student_first_name.isalpha():
        raise ValueError("The first name should not contain numbers.")
    student_last_name = input("Enter the student's last name: ")
    if not student_last_name.isalpha():
        raise ValueError("The last name should not contain numbers.")
    course_name = input("Please enter the name of the course: ")

    student: Student = Student(first_name=student_first_name, last_name=student_last_name,
                               course_name=course_name)
```

Data Classes

The Data Layer is responsible for holding the data and making sure it's structured properly. It is used for managing and storing data. It contains classes that define the data structure and manage access to that data. In OOP, these classes often represent entities or "objects" in the program, and they handle tasks such as storing, retrieving, and validating data.

```
# Create a Person Class: Data Layer
class Person:  1 usage
    """
        Represents a generic Person with first name and last name.
        Change Log:  RPathak,3/12/2025,Created Script

        """
    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name

    @property
    def first_name(self):
        return self.__first_name.title()

    """Getter method to return the first name in title case."""

    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha():  # error handling if value is not alphabets
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")
    """Setter method to ensure the first name contains only alphabetic characters."""

    @property
    def last_name(self):
        return self.__last_name.title()
    """Getter method to return the last name in title case."""

    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha():  # error handling if value is not alphabets
            self.__last_name = value
        else:
```

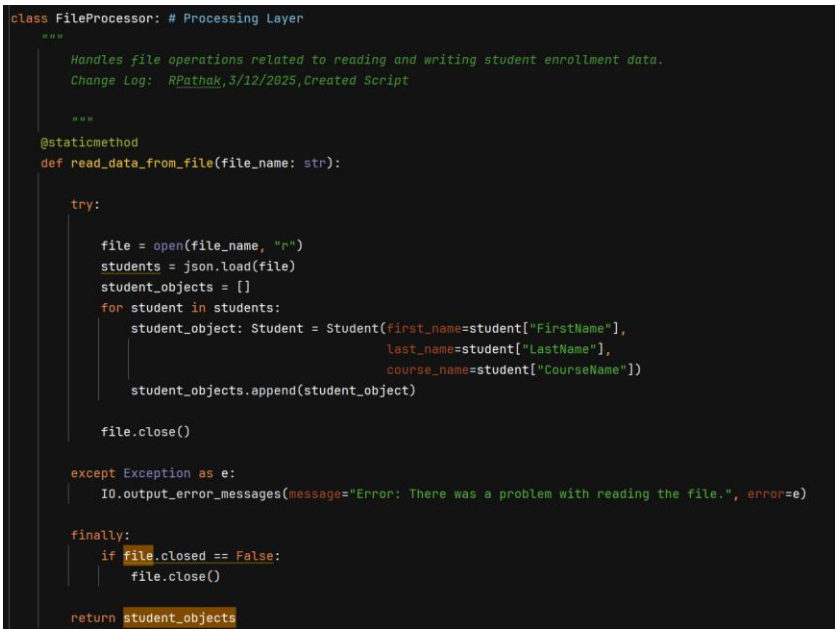
Presentation Class

A **presentation class** deals with how data is presented to the user and how the user interacts with the application. Presentation classes manages how data is displayed and how users interact with the application .It is responsible for displaying data, showing menus, accepting user input, and handling the interface. Presentation classes do not typically handle the core logic or store data; they are focused on user interaction.

```python
class IO: # Presentation layer
    """
       Handles input and output operations for the course registration program.
        Change Log:  RPathak,3/12/2025,Created Script

        """

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """
              Handles output error message and error handling
               Change Log:  RPathak,3/12/2025,Created Script
               """
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message -- ")
            print(error, error.__doc__, type(error), sep='\n')

    @staticmethod
    def output_menu(menu: str):
        """
                Shows Menu choices to user .
               Change Log:  RPathak,3/12/2025,Created Script
                """
        print()  # Adding extra space to make it look nicer.
        print(menu)
        print()  # Adding extra space to make it look nicer.
```

Processing Class

A processing class handles the business logic or operations performed on the data. It handles the logic and operations that process data. It is responsible for processing and manipulating the data, executing calculations, or interacting with external resources like databases or files. Processing classes are where the core functionality of the application takes place.



```python
class FileProcessor: # Processing Layer
    """
        Handles file operations related to reading and writing student enrollment data.
        Change Log:  RPathak,3/12/2025,Created Script

        """
    @staticmethod
    def read_data_from_file(file_name: str):

        try:

            file = open(file_name, "r")
            students = json.load(file)
            student_objects = []
            for student in students:
                student_object: Student = Student(first_name=student["FirstName"],
                                                  last_name=student["LastName"],
                                                  course_name=student["CourseName"])
                student_objects.append(student_object)

            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.closed == False:
                file.close()

        return student_objects
```

The key differences between the types of classes in an application lie in their specific responsibilities. A **data class** is focused on managing the structure and integrity of the data, ensuring that the information is stored correctly and can be accessed or modified securely. The **presentation class**, on the other hand, deals with how the data is shown to the user, handling things like displaying menus, taking user input, and presenting output. Meanwhile, the **processing class** is where the core logic of the application resides, performing operations on the data, such as calculations or interacting with external resources like files or databases. By separating these concerns into different classes, the program becomes easier to manage, maintain, and scale. Each class has a distinct role, and because they are independent, changes in one class typically won't interfere with the others, leading to more organized and flexible code.

Constructor?

A constructor is a special method in a class that is automatically called when an object of that class is created. It initializes the object's attributes with default or provided values. In Python, the constructor is defined using the __init__() method. It helps set up the initial state of an object when it's created.

```python
def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
    super().__init__(first_name=first_name, last_name=last_name)
    self.course_name = course_name
```

Attribute

An attribute is a variable that belongs to an object or class. It stores data about the object. For example, a Person class has attributes like first_name and last_name. These attributes can hold values specific to each object created from the class.

```
class Person:  1 usage
    """
        Represents a generic Person with first name and last name.
        Change Log:  RPathak,3/12/2025,Created Script

        """
    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name
```

Property

A property is a special kind of attribute in Python that allows to manage how an attribute is accessed or modified. Properties are created using the @property decorator, and they allow to add logic to control the setting or getting of values, such as validation or formatting.

```
@property
def first_name(self):
    return self.__first_name.title()

"""Getter method to return the first name in title case."""

@first_name.setter
def first_name(self, value: str):
    if value.isalpha():  # error handling if value is not alphabets
        self.__first_name = value
    else:
        raise ValueError("The first name should not contain numbers.")
"""Setter method to ensure the first name contains only alphabetic characters."""
```

Class Inheritance

Class inheritance allows a new class to take on properties and methods from an existing class. The new class, called the child class, can inherit the attributes and behavior of the parent class while also adding or modifying its own features. This helps reuse code and makes the program more organized.

```python
#  Create a Student Class; inherits from Person class above; also extends Person class by a
class Student(Person):
    """
        Represents a student enrolled in a course.
        Inherits first and last name properties from Person.
        Change Log:  RPathak,3/12/2025,Created Script

        """

    def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
        super().__init__(first_name=first_name, last_name=last_name)
        self.course_name = course_name

    @property
    def course_name(self):
        return self.__course_name

    @course_name.setter
    def course_name(self, value: str):
        self.__course_name = value

    def __str__(self):
        return f'{self.first_name},{self.last_name},{self.course_name}'
```

Overridden method?

An overridden method is a method in a child class that has the same name and parameters as a method in the parent class but is redefined with a different implementation. This allows the child class to provide its own behavior while still maintaining the same method signature.

What is the difference between Git and GitHub Desktop?

Git is a version control system that helps track changes in code over time. It allows to manage different versions of a project and collaborate with others. GitHub Desktop, on the other hand, is a graphical interface for Git that makes it easier to use for people who prefer not to work with command-line tools. It allows to manage Git repositories with a user-friendly interface.

Summary

This module explores advanced Python topics, focusing on Object-Oriented Programming (OOP) principles like inheritance and encapsulation. Python programs are structured with statements, functions, and classes, which help organize and manage code. Statements control program flow, while functions group related actions for reuse. Classes act as blueprints for creating objects that store data and perform operations. Data classes manage data integrity, presentation classes handle user interaction, and processing classes focus on core logic. Key OOP concepts such as constructors, attributes, properties, and method overriding allow for flexible and organized code. Additionally, Git is a version control system, while GitHub Desktop is a graphical interface for easier Git management.