Name: Luis Candela

Date: 11/27/2024

Course: Foundations of Programming (Python)

Module 07 – Classes and Objects

https://github.com/luiscandela87/IntroToProg-Python-Mod07

Introduction

Classes are fundamental building blocks in most programming languages, they work as modules of code organized by function, order and hierarchy. Classes are usually defined at the beginning of a script after constants and variables are introduced. Classes contain functions which are reusable blocks of code that can be referenced throughout the script without unnecessarily repeating code.
In some cases, a program needs to work with a specific part of the data in which a function must be customized to that part of the data. In Object Oriented Programming (OOP) this is called an instance class and different to the @staticmethod previously learned, an instance classes work with data and attributes together and store the methods and data in the form of in object instead of and independent list or dictionary.
Instance classes are useful in OOP for the following reasons:
Encapsulation: Code is compartmentalized into comprehensive modules of code.
Abstraction: Simplifies a program by hiding the complexity of the code and only shows the relevant information on what the program does not how it does it.
Inheritance: Subclasses pass on the properties of the “mother class” to avoid repeating code in the new classes.
In this module the student will learn the use and versatility of instance classes and how to use inheritance and abstraction to simplify code and store data as objects for a more independent manipulation of data.
We will also delve into better understanding how git hub platform works and why it has become such an important tool for program editing and collaboration.

Assignment07- Create a Program

Create a Python program that demonstrates using constants, variables, and print statements to display a message about a student's registration for a Python course. This program is very similar to Assignment06

The previous module the student course list menu, input and display codes were modified to work into several functions and grouped into @staticmethod classes depending on their role in the program. In this module we take the program a step further by converting classes into an instance-based OOP method, meaning that each function is now part of a specific instance method with specific attributes or properties.
As we will see this is very different and useful when working with real-world data because it encapsulates each student as an instance object which can be handled independently while static methods work with all students equally in list or dictionary.

Our script begins similar to the previous Module 6 program with a header, defined constants and variables.

```
# Desc: This assignment demonstrates using data classes with structured error
handling.
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   <Your Name Here>,<Date>,<Activity>
# ------------------------------------------------------------------------------------------ #

import json  # Importing json module to work with JSON data files

# Define the Data Constants
MENU: str = '''  # Defining the menu constant as a string
---- 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.
----------------------------------------
'''
```

```
FILE_NAME: str = "Enrollments.json"  # Define the file name for saving/loading
student data

# Define the Data Variables
students: list = []  # A list to hold student data (initially empty)
menu_choice: str  # Variable to hold the user's menu choice
```

The main difference in this program resides in the creation of Person “Base Class” for a generalized type of data from any kind of “Person”with a first and last name. The Student Subclass inherits the properties from the Person Class but also handles specific attributes of a student like the course name.
Using the Person Class as an example we will explain each step of its construction:

1. The class begins with a constructor (__init__) that defines the variables where the instance will store the new data. In this case first and last name.
2. A @property is a decorator that indicates the call of a “getter method” to get the value of each attribute and then a “setter method” to evaluate its content.
3. __str__ is a method to return a printed version of the attribute values

```
# Define the Person Class (Base Class)
class Person:
    # A class that represents a person with a first name and last name.

    def __init__(self, first_name: str = '', last_name: str = ''):
        # Initialize the Person object with first name and last name (no course_name
here).
        Self.__first_name = first_name
        self.__last_name = last_name

    @property
    def first_name(self):
        # Getter for first_name property, returns the title-cased first name.
        return self.__first_name.title()

    @first_name.setter
```

```
    def first_name(self, value: str):
        # Setter for first_name property, ensures value is alphabetic or empty.
        If value.isalpha() or value == “”:
            self.__first_name = value
        else:
            raise ValueError(“The first name should not contain numbers.”)

    @property
    def last_name(self):
        # Getter for last_name property, returns the title-cased last name.
        return self.__last_name.title()

    @last_name.setter
    def last_name(self, value: str):
        # Setter for last_name property, ensures value is alphabetic or empty.
        If value.isalpha() or value == “”:
            self.__last_name = value
        else:
            raise ValueError(“The last name should not contain numbers.”)

    def __str__(self):
        # Override the __str__() method to return a formatted string representation
of the Person.
        Return f’{self.first_name} {self.last_name}’  # Returning the person’s full
name
```

Since we are talking about student data the program creates a Student Sublass that inherits properties from the Person Class using a super() constructor method. It calls the parent class’s attributes and overrides it adding a new attribute string for the course name and finishes by saving it as a new object.

```
# Define the Student Class (Derived Class from Person)
class Student(Person):
    # A subclass that represents a student. Inherits from Person class.
```

```
    # Adds functionality specific to students, such as course name.

    def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
        # Initialize the Student object with first name, last name, and course name.
        super().__init__(first_name=first_name, last_name=last_name)  # Call the
parent constructor
        self.__course_name = course_name  # Now only here in the Student class.

    @property
    def course_name(self):
        # Getter for course_name property, returns the course name.
        return self.__course_name

    @course_name.setter
    def course_name(self, value: str):
        # Setter for course_name property, ensures it's a valid string.
        self.__course_name = value

    def __str__(self):
        # Override __str__() method to include course information.
        return f'{super().__str__()} is enrolled in {self.course_name}'  # Calls Person's
__str__ and adds course info
```

Once the instance classes are defined the program proceeds to define the @staticmethods that work with the files and user input like Module 06 version. The following modifications to the code will change how to work with the new objects created:

1. FileProcessor Method reads the JSON file list of dictionaries and converts it to a list of objects, appends new data and save it again in the original format.

```
# Processing Layer (File handling and data loading/saving)
class FileProcessor:
    # A class to process reading and writing student data from/to a JSON file.

    @staticmethod
```

```
    def read_data_from_file(file_name: str, student_data: list):
        # Reads data from a JSON file and loads it into a list of dictionaries.
        Try:
            with open(file_name, "r") as file:
                student_dicts = json.load(file)  # Load data from the file as a list of dictionaries
                # Convert dictionaries back to Student objects
                for student_dict in student_dicts:
                    student = Student(
                        first_name=student_dict["FirstName"],
                        last_name=student_dict["LastName"],
                        course_name=student_dict["CourseName"]
                    )
                    student_data.append(student)
        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
        return student_data

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        # Writes student data to a JSON file.
        Try:
            # Convert Student objects to dictionaries before saving
            student_dicts = [{
                "FirstName": student.first_name,
                "LastName": student.last_name,
                "CourseName": student.course_name
            } for student in student_data]
            with open(file_name, "w") as file:
                json.dump(student_dicts, file)  # Save list of dictionaries
            IO.output_student_and_course_names(student_data)  # Display saved data
        except Exception as e:
            message = "Error: There was a problem with writing to the file.\nPlease check that the file is not open by another program."
            IO.output_error_messages(message=message, error=e)
```

2. The IO class is responsible for user interaction (input/output), and it interacts with Person and Student objects in terms of displaying and inputting student data. Data output refers the Student " __str__ method" to display the student information.

```
# Presentation Layer (Handling user input and output)
class IO:
    # A collection of functions that manage user input/output.

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        # Displays error messages to the user.
        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):
        # Displays the main menu to the user.
        print()  # Adding extra space to make it look nicer.
        print(menu)
        print()  # Adding extra space to make it look nicer.

    @staticmethod
    def input_menu_choice():
        # Prompts the user to enter a menu choice.
        choice = "0"
        try:
            choice = input("Enter your menu choice number: ")
            if choice not in ("1", "2", "3", "4"):  # Ensure valid choice
                raise Exception("Please, choose only 1, 2, 3, or 4")
        except Exception as e:
            IO.output_error_messages(e.__str__())  # Display error message
```

```
        return choice

    @staticmethod
    def output_student_courses(student_data: list):
        # Displays the names and courses of all registered students.
        print("-" * 50)
        for student in student_data:
            print(student)  # Automatically calls the __str__ method of the Student
class
        print("-" * 50)
```

3. The input obtains new information from the user and converts it to an object and appends it to the list.
4. Error handling continues to prompt the user for data type errors or missing files.

```
    @staticmethod
    def input_student_data(student_data: list):
        # Prompts the user for a student's details and adds them to the student_data
list.
        try:
            # Get student details
            student_first_name = input("Enter the student's first name: ")
            if not student_first_name.isalpha():
                raise ValueError("First name should only contain letters.")
            student_last_name = input("Enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError("Last name should only contain letters.")
            course_name = input("Please enter the name of the course: ")

            # Create a Student object and add it to the list
            student = Student(first_name=student_first_name,
last_name=student_last_name, course_name=course_name)
            student_data.append(student)
            print(f"You have registered {student_first_name} {student_last_name} for
{course_name}.")
        except ValueError as e:
```

```
            IO.output_error_messages(message="Input data is not valid!", error=e)
        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with
your entered data.", error=e)
        return student_data
```

5. The main body remains similar to the original version although it now works with objects which are manipulated by functions which refer to the classes.

```
# Main Program Execution

# When the program starts, read the file data into a list of students
students = FileProcessor.read_data_from_file(file_name=FILE_NAME,
student_data=students)

# Main program loop
while True:
    # Show the menu to the user
    IO.output_menu(menu=MENU)

    # Get the user's menu choice
    menu_choice = IO.input_menu_choice()

    # Handle user choice
    if menu_choice == "1":
        # Register a new student
        students = IO.input_student_data(student_data=students)
        continue
    elif menu_choice == "2":
        # Show current student data
        IO.output_student_courses(students)
        continue
    elif menu_choice == "3":
        # Save data to file
        FileProcessor.write_data_to_file(file_name=FILE_NAME,
student_data=students)
        continue
```

```
    elif menu_choice == "4":
        # Exit the program
        break
    else:
        print("Please only choose option 1, 2, 3, or 4")

print("Program Ended")  # Indicate program termination
```

Summary

This module highlights the importance of Object-Oriented Programming (OOP) principles like encapsulation, abstraction, and inheritance in structuring efficient, maintainable code. By transitioning from static methods to instance-based methods, we can better handle real-world data through objects with specific attributes and behaviors, reducing code redundancy and improving flexibility. Encapsulation protects an object's internal state, while abstraction simplifies complex systems by hiding implementation details. Inheritance allows for code reuse, enabling subclasses to inherit properties and methods from parent classes. Additionally, error handling techniques, such as safely managing file reading operations, help prevent data loss and ensure program robustness.
Finally GitHub is a web-based platform that provides version control using Git, a distributed version control system. It allows developers to store, manage, and track changes to their code repositories over time. GitHub enables collaboration among developers, making it easy for multiple people to work on the same project simultaneously, track modifications, and merge contributions.