Rabiya Wasiq

11/27/2023

Introduction to Programming with Python

Assignment07

CLASSES AND OBJECTS

Introduction

This week we went further into classes; we learned about Data Class and its components. We learned about constructors and working with The Self Keyword. We were able to work with Class properties, also private properties. We learned the getter and setter methods. We also learned about inheritance class and how the sub-class would inherit properties from the parent-class. We learned Python's magic methods namely the __init__ and __str__ method. We also learned to use GIT from our Desktop and also Pycharm.

Creating Python Script

For the purpose of this assignment, I will be using my code from Assignment 06 as the starting point. I will then be adding a data class Person and then a subsequent sub-class Student. We will add data validation to class properties, and I will also be demonstrating overriding methods.

Data Classes

Data class is a class that is designed to only hold data values, we can create instances of data classes to create objects by passing in arguments to class attributes. While processing and presentation classes usually just have methods, Data classes typically have Attributes, Constructors, Properties, in addition to Methods.

Attributes

In programming, an attribute is a piece of data, or a characteristic associated with an object. Attributes describe or store information about the object they belong to. Depending on the programming context, attributes are also referred by developers as fields. Attributes are variables that hold data specific to an object. These variables can have different data types, such as integers, strings, or custom data types.

Constructor

Constructors are generally used for instantiating an object. The task of constructors is to initialize (assign values) to the data members of the class when an object (instance) of the class is created. In Python the __init__() method is called the constructor and is always called when an object is created.

Properties

Properties are functions designed to manage attribute data. Typically, for each attribute, you create two properties—one for "getting" data and one for "setting" data. These functions are commonly known as "Getters" and "Setters" or more formally as "Accessors" and "Mutators."

Starter Code from Assignment06

We start by using Open Tab in Pycharm and navigating to the location of Python script of Assignment 06. We will copy the script and make changes as we proceed.

We will use the same format of adding a script header and using pseudocode for problem solving.

```
# ------------------------------------------------------------------------
-------------- #
# Title: Assignment07
# Desc: This Assignment Demonstrates classes and objects
# Change Log: (Who, When, What)
#   Rabiya Wasiq,11/26/2023,Created Script
#   Rabiya Wasiq 11/27/2023, Updated comments
# ------------------------------------------------------------------------
--------------- #
```

Figure 1 – Script Header

Class : Person

I have defined a class Person, my class has two attributes first_name and last_name, which I have initialized using the __init__ constructor. The "__" before first_name and last_name indicates that these are private properties and should not be accessed outside of the class. **(Figure 2)**

```
#-----------Data classes-------------
class Person:
    """
    A class representing person data.

    Properties:
    - first_name (str): The student's first name.
    - last_name (str): The student's last name.

    ChangeLog:
    - Rabiya Wasiq, 11.26.2030: Created the class.
    """
```

```
    def __init__(self, first_name: str = '', last_name: str = ''):
        self.__first_name = first_name
        self.__last_name = last_name
```
Figure 2 – Class Person

I have added the getter property function to my first_name along with additional formatting using .capitalize(). This helps protect my private attributes outside of the class and returns the first_name as capitalized.
The setter function is also a tool to protect your private attributes by way of data validation. I have added data validation to my setter function using if condition. The value passed in for .first_name will only be set as the value for self.__first_name if the condition is met. Else it will raise a Value Error. **(Figure 3)**

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

@first_name.setter
def first_name(self, value: str):
    if value.isalpha() or value == "":  # is character or empty string
        self.__first_name = value
    else:
        raise ValueError("The first name should not contain numbers.")
```
Figure 3 – Getter and Setter property function for first_name

I have followed the same pattern for Person class attribute last_name.**(Figure 4)**

```
@property
def last_name(self):
    return self.__last_name.capitalize()  # formatting code

@last_name.setter
def last_name(self, value: str):
    if value.isalpha() or value == "":  # is character or empty string
        self.__last_name = value
    else:
        raise ValueError("The last name should not contain numbers.")
```
Figure 4 – Getter and Setter property function for last_name

String Method (Magic Method)

Magic methods simplify common tasks in Python programming. For instance, by defining __str__, you can easily control the string representation of an object, making it more readable and user-friendly.

Here I have defined the string representation for my class Person. **(Figure 5)**

```
def __str__(self):
    return f'{self.first_name},{self.last_name}'
```
Figure 5 – String representation for class Person

Sub – class : Student

I will now create a sub- class Student that will inherit the properties from the parent class. We will be using the overloading method to add more attributes.

Class Student(Person) inherits the attributes (first_name and last_name) from the parent class using the super.__init__ function. We are using the overloading method to add another attribute course_name to our student class. **(Figure 6)**

```
class Student(Person):
    """
    A sub-class of Person
    A class representing student data.

    Properties:
    - first_name (str): The student's first name.
    - last_name (str): The student's last name.
    - course_name: The course registered for by the student.

    ChangeLog:
    - Rabiya Wasiq, 11.26.2030: Created the class.
    """

    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
```

Figure 6 – Student class (sub-class of Person class)

Since the getter and setter are already defined in the parent class , I only need to define them for the attribute course_name **(Figure 7)**

```
@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} '
```

Figure 7 – Setter and Getter for course_name attribute for Student Class

I have used the over riding method to the string representation for Student Class.I will use this to present data to the user. **(Figure 7)**

Class : FileProcessor

Moving forward we need to make changes to our read_data_from_file function.

```
class FileProcessor:
    """
    A collection of processing layer functions that read and write data from
file
```

```
    Rabiya Wasiq, Created class, 11/19/23
    """

    @staticmethod
    def read_data_from_file(File_Name: str, student_data: list[Student]) ->
list[Student]:
        """
        This function reads data from Json file and stores it into a list of
dictionaries
        :param File_Name:
        :return: list[dict[str.str,str]]
            Rabiya Wasiq, 11/19/23, Created Function
        """
        File_Name : str
        list_of_dictionary_data : list[dict[str,str,str]] = []
        file :TextIO = None
        try:
            file = open(File_Name, 'r')
            list_of_dictionary_data = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_message('Json file not found, creating it...',e)
            file = open(File_Name, 'w')
        except JSONDecodeError as e:
            IO.output_error_message('Json file does not contain any data,
resetting it..',e)
            file = open(File_Name, 'w')
            json.dump(list_of_dictionary_data, file)
        except Exception as e:
            IO.output_error_message('Unexpected Technical error',e)
        finally:
            if not file.closed:
                file.close()
        for student in list_of_dictionary_data:
            student_object: Student =
Student(first_name=student["First_Name"],
                                        last_name=student["Last_Name"],
                                        course_name=student["Course_Name"])
            student_data.append(student_object)
        return student_data
```

Figure 8 – Read_data_from_file function

In **(Figure 8),** you can see that most of our code is the same as Assignment 06, apart some changes. Our Json file currently holds data in the form of a list of dictionaries. So, we read the data and store in a local variable list_of_dictionary_data. After that we iterate over each dictionary using the for loop and pass the values to our student_object (an instance of student class). In the end we want our function to return list of students.

We also need to update the function writing_data_to_file **(Figure 9)**

```
@staticmethod
def writing_data_to_file(new_student: Student, student_data:list[Student],
File_Name: str ):
    """
```

```
    Writes data to Json file in the format list of dictionaries.
    :param student_row:
    :param students_data:
    :return:None
            Rabiya Wasiq, 11/19/23, Created Function
    """
    student_data: list[Student]
    File_Name: str
    file: TextIO = None
    list_of_dictionary_data: list[dict] = []
    new_student: Student

    if new_student.first_name == '' or new_student.last_name == '' or
new_student.course_name == '':
        IO.output_message('Please enter student details')

    else:
        for student in student_data:
            student_json = {"First_Name": student.first_name, "Last_Name":
student.last_name, "Course_Name": student.course_name}
            list_of_dictionary_data.append(student_json)

        try:
            file = open(File_Name, 'w')  # using the write function, to
truncate the file
            json.dump(list_of_dictionary_data, file)
            file.close()
            IO.output_message('Student registration details recorded\n')
        except Exception as e:
            IO.output_error_message('Unexpected Technical error',e)
        finally:
            if not file.closed:
                file.close()
```

Figure 9 – Write_data_to_file

Since we now have data stored as a list of students, while writing data to Json file we need to convert it back to a list of dictionaries. We do this by again using a for loop and iterating over each student (object)

Class : IO

For our Class IO the functions output_error_message, output_message, output_menu and input_menu_choice remain the same.

The function current_data_from_file presents the student data from Json file to the user. I have used the for loop to iterate over the list of students and used string representation of Student class to present data to the user **(Figure 10)**

```
@staticmethod
def current_data_from_file(student_data:list[Student]) ->str:
    """
    This function displays all student data from the Json file, formatted in
a string
    :param student_data:
```

```
    :return:str
            Rabiya Wasiq, 11/19/23, Created Function
    """
    student_data: list[Student]
    for student in student_data:
        IO.output_message(str(student))
```

Figure 10 – current_data_from_file

We will also need to update the input_student_data function. Since we now have data validation set up in our setter .first_name and .last_name we do not need it here. The function appends the new student object to our list of students and also returns the new student. **(Figure 11)**

```
@staticmethod
def input_student_data(student_data : list [Student]) -> Student:
    """
    This function gets first name, last name, course name from the user and
adds them to a dictionary
    :param student_data:
    :return: Student
            Rabiya Wasiq, 11/19/23, Created Function
    """
    student_data: list[Student]
    student = Student()
    while True:
        try:
            student.first_name = input("What is the student's first name? ")
            break
        except ValueError:
            IO.output_error_message('Student First Name can only contain
alphabetic characters')
            #Not passing error deatils
    while True:
        try:
            student.last_name = input("Enter the student's last name: ")
            break
        except ValueError:
            IO.output_message('Student Last Name can only contain alphabetic
characters')

    student.course_name = input("Enter the course name: ")
    student_data.append(student)
    return student
```

Figure 11 – Input_student_data

To present student details received from the user, we use the present_student_data function, using the getter function. (**Figure 12)**

```
@staticmethod
def present_student_data(new_student : Student):
    """
    This function presents data from a dictionary to the user in string
formatting
    :param student_row:
```

```
    :return:None
            Rabiya Wasiq, 11/19/23, Created Function
    """
    new_student:Student
    if new_student.first_name == '' or new_student.last_name == '' or
new_student.course_name == '':
        IO.output_message('Please enter student details again')
    else:
        message = f'{new_student.first_name} {new_student.last_name} has
registered for {new_student.course_name}'
        IO.output_message(message)
```

Figure 12 - present_student_data

The function exit choice remains unchanged.

Declaring and assigning Variables / Constants

Declaring and assigning variables are the first steps to writing a code. It is best practice to declare the variables and constants that you intend to use throughout the script at the beginning. I have declared my variables before the main body of my code after I defined my classes.**(Figure 13)**

```
# Define the Data Constants
MENU: str = '''
-----------------------------------------
---- Course Registration Program ----
  Select from the following menu:
    1. View all students registered to date
    2. Register a New Student for a Course
    3. Show New student registration details
    4. Save New student data to a file
    5. Exit the program

-----------------------------------------
'''

# Define the Data Variables
FILENAME: str = 'Enrollments.json'
menu_choice: str = ''
new_student: Student = Student()
students: list[Student] = []
```

Figure 13: Declaring and assigning constants and variables.

Main body of the code

After defining the functions at the start of the code, we can now move on to the main body of the code. We will execute our code by calling the functions that we have defined and pass in the relevant arguments. We will be storing the return values to code's variables as we progress.

```
# Present and Process the data
while True:

    IO.output_menu(menu=MENU)   # Present Menu
```

```
    menu_choice = IO.input_menu_choice()

    # Menu choice 1 shows the data extracted from the JSON and saved in the
two-dimensional list
    if menu_choice == '1':
        students =
FileProcessor.read_data_from_file(File_Name=FILENAME,student_data=students)
        IO.current_data_from_file(student_data=students)

    # Getting student details from the user
    elif menu_choice =='2':
        new_student = IO.input_student_data(student_data=students) #New
student object created

    #presenting new student registration details to the user
    elif menu_choice =='3':
        IO.present_student_data(new_student=new_student)

    #writing new student details to Json file
    elif menu_choice =='4':
        FileProcessor.writing_data_to_file(new_student
=new_student,student_data=students,File_Name=FILENAME)

    #exiting the program
    elif menu_choice == '5':
        exit_choice = IO.exit_choice()
        if exit_choice == "Y":
            IO.output_message('\nPausing the program till you press
Enter...\n')
            break
```

Figure 13 – While Loop

I then start a while loop that would prompt the menu to the user using the IO.output_menu function and passing in the constant MENU and stores the return value in a variable menu_choice.

I then start adding my if conditions.

If menu_choice is 1, I present the data read from the JSON file to the user using the IO.current_data_from_file function and passing in the variable "students".

If menu_choice is 2, I prompt the user to enter details using the IO.input_student_data and store the return value in the form of a student object "new_student".

If menu_choice is 3, I present the student details received using the IO.present_student_data and passing in the variable "new_student" (Student object from menu_choice 2). This function also appends our list of students with the new_student

If menu_choice is 4, I write the student details received to a Json file, using the FileProcessor.writing_data_to_file function.

If menu_choice is 5, I present the user with the choice to exit the program using the IO.exit_choice function

Summary

I was successfully able to create a code that demonstrates the use of data classes, constructors and properties. I was also able to create sub-classes using inheritance, overloading and over riding. The data class attributes made it much easier to access values and reduced the chances of human error by way of adding data validation in the property setter function.