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.cours***e\_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.