Alexander Clark

27Nov2023

IT FDN 110 B

Assignment06

https://github.com/arclark360/IntroToProg-Python-Mod07/

MOD 7 Classes and Objects

Introduction

This week in Module 7 we took a closer look at Python Classes and got a better understanding of object-oriented programming. We looked at the constructor, properties, and inheritance aspects of Classes and then used this new knowledge in developing our existing projects.

Attributes of Classes

Constructors

Previously, we only used classes to group specific types of functions, such as **Class IO** and **Class File Processing**. While this is still a way of modulating the code, taking it a step further involves giving a class **parameters** that it can store. For example, we could create the Class Student to store information about the student, such as Name, Enrolled Classes, GPA, etc. This allows you to create instances of these classes and work with data in the form of objects.

In Python, a **constructor** is a method that is automatically called when an object or class is created, and it is used to initialize the object's attributes. In **Figure 1**, an example code is given to illustrate the syntax of the constructor method. It starts with a unique format, '**__init__(self,x,y)'**, enclosed in double underscores on both sides. Inside the parentheses, you provide the attributes associated with the parameters and a default self parameter.

In **Figure 1**, there is a class called **MyClass** with two parameters. An instance of the class, '**my_object'**, is created with each parameter filled in. Then, '**my_object'** is manipulated using the attributes specified during the object's construction.

```
1 usage
class MyClass:
    def __init__(self, parameter1, parameter2):
        # This method is automatically called when an object is created
        self.attribute1 = parameter1
        self.attribute2 = parameter2

# Creating an object of MyClass
my_object = MyClass( parameter1: "value1", parameter2: "value2")

# Accessing the attributes of the object
print(my_object.attribute1)  # Output: value1
print(my_object.attribute2)  # Output: value2
```

Figure 1. Example of a Class with attributes

In our Assignment this week we will use this new idea of Classes with parameters to create a Student and Person Class for the student registration program.

Properties & Getters

After learning about attributes and constructors, we delved into the concept of properties, which enables the control of access, modification, and deletion of attributes in a class. Properties allow you to define getter, setter, and deleter methods for an attribute. The rationale behind using properties is that sometimes you want to keep the data that the attributes represent private and not acted upon directly. In Python, an attribute with an '_' in front of its name is typically considered private.

Using a **getter**, you can define a public version of a method to retrieve the data instead of accessing the private version directly. Figure 2 illustrates this concept, where the **@property** decorator initiates the syntax for the getter. The method is then defined without the '_' and returns the attribute value.

Figure 2 also portrays a setter, which is used to modify an attribute. In this example, it capitalizes the return value. The setter instead uses the **attributes name and .setter** in the syntax instead of the @property of the getter.

```
class MyClass:
    def __init__(self):
        self._my_attribute = None

    1 usage
    @property
    def my_attribute(self):
        return self._my_attribute

    @my_attribute.setter
    def my_attribute(self, value):
        self._my_attribute = value.capitalize
```

Figure 2. Getters and Setters

Polymorphism & Inheritance

The last concept we covered for the week was polymorphism and the concept of inheritance in relation to classes. This refers to Python's ability to allow different objects to be used interchangeably and to treat objects of different types as objects of a common type. For example, you might have a class '**Animal'** with its own properties and then have subclasses underneath it, like '**Dog'** or '**Cat'**, that inherit some attributes from the parent class. This saves on the syntax of having to rewrite the attributes for each subclass.

Figure 3 illustrates an example from the homework assignment. In this example, the class '**Student'** is a subclass of the '**Person'** class. Instead of using '**self.'** to create the attributes for first name and last name, the '**super()'** command is used to initialize the creation of a person with the first name and last name. Since a Student is a Person, and a Person always, in this code, has a name.

```
10 usages
class Student(Person):
    def __init__(self, first_name: str, last_name: str, course_name: str) -> None:
        super().__init__(first_name=first_name, last_name=last_name)
        self._course_name = course_name
```

Figure 3. Example of a Subclass

Assignment

Overview

In this week's assignment we take the student registration program and adding in the Classes of Student and Person. Once we add in the new classes we will look at ways in which we can adjust the code to handle the modifications.

Creating Student and Person

When creating the '**Student'** and '**Person'** classes, I decided to start with the Person class and work downward to the Student subclass. **Figure 4** depicts the Person class, which has two parameters: **'first_name'** and '**last_name'**. For both attributes, I made them private and created getter and setter properties. Both properties are designed to capitalize the data to ensure there are no lowercase names. I also added a check to verify that the name is a string.

Once the Person class was complete, I proceeded to create the Student class, as illustrated in **Figure 5**. This class was a bit easier since I could establish a relationship with 'Person' by making 'Person' a parameter of 'Student' and using the **super()** command to ensure that a Person is created every time a Student is created. For the Student class, I added the parameter '**course_name'** and implemented getter and setter functions for it. Additionally, I included a '**__str__** 'method which will be useful in code modification in choice 3.

With the classes created, the next step was to integrate them into the code.

```python
class Person:
    def __init__(self, first_name: str, last_name: str) -> None:
        self._first_name = first_name
        self._last_name = last_name

    def __str__(self) -> str:
        return f"This is {self.first_name} {self.last_name}"

    2 usages
    @property
    def first_name(self):
        return self._first_name.capitalize()

    1 usage
    @first_name.setter
    def first_name(self, value):
        if value.isalpha():
            self.first_name = value
        else:
            raise ValueError("first name must be alphabetic")

    2 usages
    @property
    def last_name(self):
        return self._last_name.capitalize()

    1 usage
    @last_name.setter
    def last_name(self, value):
        if value.isalpha():
            self.last_name = value
        else:
            raise ValueError("first name must be alphabetic")
```

Figure 4. Person Class

```python
class Student(Person):
    def __init__(self, first_name: str, last_name: str, course_name: str) -> None:
        super().__init__(first_name=first_name, last_name=last_name)
        self._course_name = course_name

    4 usages
    @property
    def course_name(self):
        return self._course_name

    @course_name.setter
    def course_name(self, value):
        if value.isalpha():
            self.last_name = value
        else:
            raise ValueError("first name must be alphabetic")

    def __str__(self) -> str:
        return f"You have registered {self.first_name} {self.last_name} for {self.course_name}."
```

Figure 5. Student Class

Code Modification

To implement the Student class into the code, I had to analyze the program and replace all instances where the program was expecting a **list[dict[str,str,str]]** and replace it with **list[Student]**. So now instead of operating with a list of dictionaries, the program now works with a list of Student objects. In most cases, this involved changing the call from a specific key parameter of the dictionary to a specific attribute of the Student class object being worked with.

One area where this change was challenging was the read and write functions of the program, due to the fact JSON files and commands cannot directly work with custom Student objects. **Figure 6** depicts how the program initially reads the JSON file and stores it in the previously used format of dictionaries. Once everything is read, a '**for'** loop is used to iterate over the matrix, creating a Student for each row of data. These Student objects are then stored in a **list[Student],** which is returned at the end of the process. This method is similar when writing to the JSON, but in reverse. When writing to the file, you must create a list of dictionaries based on the list of Student objects and then write the list of dictionaries to the JSON.

```python
def read_data_from_file(file_name: str) -> list[Student]:
    """
    Reads the stored JSON file into the application and stores it as a list of dictionaries in students
    :param file_name: The file name that the program will read
    :param students_matrix: The students list of dictionaries
    :return: The updated students list of dictionaries
    """
    file = TextIO
    json_data: list[dict[str, str, str]] = []
    while True:
        try:
            file = open(file_name, "r")
            json_data += json.load(file)
        except FileNotFoundError as e:
            IO.output_error_messages(message="JSON File not found, creating JSON file", error=e)
            file = open(file_name, "w")
            json.dump(json_data, file)
        except json.JSONDecodeError as e:
            IO.output_error_messages(
                message="File is not in right format and can't load students, please review file for error",
                error=e)
            break
        except Exception as e:
            IO.output_error_messages(message="Undefined Error", error=e)
            break
        finally:
            file.close()

        student_matrix: list[Student] = []
        for row in json_data:
            student_matrix.append(Student(row['first_name'], row['last_name'], row['course_name']))
        return student_matrix
```

Figure 6. read_data_from_file utilizing the Student Class

One area where we were able to take advantage of the new Student Class was in terms of the **"__str__"** attribute that is used for Students. Were able to use a for loop and the method to print out the unique f string we used to have for choice 3. **Figure 7** depicts this modification.

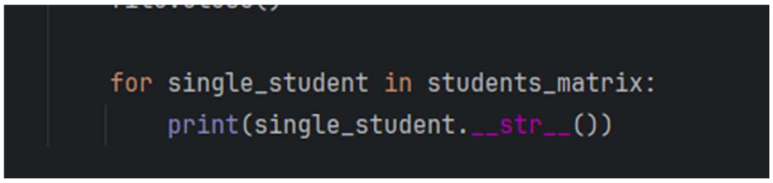

Figure 7. Menu Choice 3 modification for printing saved data.

Summary

Overall this week, we enhanced our comprehension of Object-Oriented Programming by delving deeper into the Class element of Python. We explored the intricacies of creating instances of objects with specific attributes and learned how to effectively modify these attributes. This newfound knowledge proved invaluable as we seamlessly integrated it into our student registration program, introducing the Student and Person classes.

Seeing the evolution of our original program to its current state is truly satisfying. As we reflect on this point in the course, I eagerly anticipate uncovering additional possibilities to enhance and refine our program further.

Test Output

Once all the modifications were made the code was ready to be tested:

Figure 8. PyCharm

```
What would you like to do: 1
Enter the student's first name: Jeff
Enter the student's last name: Shu
Please enter the name of the course: python 2

---- 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.
----------------------------------------

What would you like to do: 2

The current data is:
List Data: [<__main__.Student object at 0x0000026E
String Format:
Alex, Clark, python 1
Jeff, Shu, python 2

---- 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.
----------------------------------------

What would you like to do: 3
You have registered Alex Clark for python 1.
You have registered Jeff Shu for python 2.

---- 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.
----------------------------------------
```

```
What would you like to do: 4
Program Ended

Process finished with exit code 0
```

Figure 9. Command Prompt

```
PS C:\Users\arcla\OneDrive\Desktop\Python Mods\

---- 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.
----------------------------------------

What would you like to do: 1
Enter the student's first name: Feon
Enter the student's last name: Shu
Please enter the name of the course: python 3

---- 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.
----------------------------------------

What would you like to do: 2

The current data is:
List Data: [<__main__.Student object at 0x00000
String Format:
Alex, Clark, python 1
Jeff, Shu, python 2
Feon, Shu, python 3

---- 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.
----------------------------------------

What would you like to do: 3
```

```
You have registered Alex Clark for python 1.
You have registered Jeff Shu for python 2.
You have registered Feon Shu for python 3.

---- 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.
----------------------------------------

What would you like to do: 4
Program Ended
PS C:\Users\arcla\OneDrive\Desktop\Python Mods\_Module07\Ass
```