Name: Nelida Romo Tijerina
Date: November 24, 2023
Course: IT FDN 110 B Au 23: Foundations of Programming: Python

Assignment 07 – Classes and Objects

Introduction

The objective of this assignment is to 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, but this assignment demonstrates how to create and use classes to manage data and with structured error handling

This document includes a description of the steps that I followed while learning about objects, classes, inheritance and Python magic methods such as __str__() "to string" method.

1. Person and Student Classes

A. Person Class:

In this part, in relation with the previous Assignment06, I only added the following two classes: Person Class (the parent of the Student Subclass of Person) and Student Class (Subclass of Person).

From the Mod07-Notes that we have from class:

- Person Class (Superclass of Student): The Person class is defined with a constructor (__init__) that takes two optional parameters: first_name and last_name. It initializes the first_name and last_name attributes with the values provided during object creation. The __str__ method is defined to return a string representation of a Person object, combining the first_name and last_name attributes.

Figure 1 and **Figure 2** illustrate my code for this Person Class.

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

    4 usages (2 dynamic)
    @property
    def first_name(self):
        return self._first_name.title()  # formatting code

    3 usages (2 dynamic)
    @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.")

    4 usages (2 dynamic)
    @property
    def last_name(self):
        return self._last_name.title()  # formatting code

    3 usages (2 dynamic)
    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha() or value == "":  # is character or empty string
```

Figure 1. Person Class: Getters and Setters: Part 1.

```python
    def first_name(self):
        return self._first_name.title()  # formatting code

    3 usages (2 dynamic)
    @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.")

    4 usages (2 dynamic)
    @property
    def last_name(self):
        return self._last_name.title()  # formatting code

    3 usages (2 dynamic)
    @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.")

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

Person › __init__()

Figure 2. Person Class: Getters and Setters: Part 2.

B. Student Class:

- Student Class (Subclass of Person): The Student class is defined as a subclass of the Person class by specifying Person within parentheses after the class name. This indicates explicit inheritance, which means that Student inherits all attributes and methods from Person.

Figure 3 illustrates my code for this Student Class.

```python
    - first_name (str): The student's first name.
    - last_name (str): The student's last name.
    - course_name (str): The course name of the student.

    ChangeLog:
    - Nelly,22.11.2023,Created 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

    4 usages (2 dynamic)
    @property
    def course_name(self):
        return self._course_name

    3 usages (2 dynamic)
    @course_name.setter
    def course_name(self, value: str):
        if isinstance(value, str):  # Assuming course name can be any string
            self._course_name = value
        else:
            raise ValueError("Course name must be a string.")

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

Student
```

Figure 3. Student Class

2. <u>FileProcessor Class</u>

A. Read Data from file.

For this function, something that was very useful for me was to ask myself, what I have for this function and what I need as a result of this being executed.

When reading data from the file, what you have are dictionaries and should create Student objects from the loaded data and add them to “students” list.

I have included below a screenshot that illustrates my code for reading data from file function. **Figure 4**.

.

```python
@staticmethod
def read_data_from_file(file_name: str):
    """ Reads data from a JSON file and loads it into a list of Student objects

    ChangeLog: (Who, When, What)
    Nelly,22.11.2023,Created function

    :param file_name: string data with name of file to read from

    :return: list of Student objects
    """
    students = []
    try:
        with open(file_name, "r") as file:
            student_dicts = json.load(file)
            for student_dict in student_dicts:
                students.append(Student(student_dict["FirstName"],
                                        student_dict["LastName"],
                                        student_dict["CourseName"]))
    except FileNotFoundError:
        print("File not found, starting with an empty list of students.")
    except Exception as e:
        IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
    return students
```

Figure 4. Read Data from File.

B. Write Data to file.

In this part what we need is to convert ‘Student’ objects back into dictionaries that can be written to a JSON file.

I have included below a screenshot of my code for this part.

```python
def write_data_to_file(file_name: str, student_objs: list):
    """ Writes a list of Student objects to a JSON file

    ChangeLog: (Who, When, What)
    Nelly,23.11.2023,Created function

    :param file_name: string data with name of file to write to
    :param student_objs: list of Student objects to be writen to the file

    :return: None
    """

    try:
        with open(file_name, "w") as file:
            student_dicts = []
            for student in student_objs:
                student_dicts.append({
                    "FirstName": student.first_name,
                    "LastName": student.last_name,
                    "CourseName": student.course_name
                })
            json.dump(student_dicts, file)
    except Exception as e:
        message = "Error: There was a problem with writing to the file.\n"
        message += "Check that the file is not open by another program."
        IO.output_error_messages(message=message,error=e)
```

Figure 5. Write Data to File

3. Class I/O

Here, the key was to modify IO.input_student_data() to create ‘Student’ objects and modify IO.output_student_and_course_name() to work with ‘Student’ objects.

A. Output student and course name

This function displays the student and course names to the user. As you can see in **Figure 6** line 226, the parameter to use is student_objs, that is the list of Student objects to be displayed.

```python
    @staticmethod
    def output_student_and_course_names(student_objs: list):
        """ Displays the student and course names to the user

        ChangeLog: (Who, When, What)
        Nelly,22.11.2023,Created function

        :param student_objs: list of Student objects to be displayed

        :return: None
        """

        print("-" * 50)
        for student in student_objs:
            print(f'Student {student.first_name} {student.last_name} is enrolled in {student.course_name}')
        print("-" * 50)

```

Figure 6. output_student_and_course_names function

B. Input student data

As shown in **Figure 7** this function gets data from the user and returns a ‘Student’ object, line 252.

Also, in line 249 you can see how the data is sent to the student list of objects in this case.

What happens in detail here (line 249) is:

-Student refers to the Student class, which is defined to inherit from the Person class.

-The Student class has an __init__ method that accepts three parameters: first_name, last_name, and course_name.

-When Student(student_first_name, student_last_name, course_name) is called, Python creates a new object in memory of type Student.

-It then calls the __init__ method of the Student class, passing the given student_first_name, student_last_name, and course_name as arguments.

-Inside the __init__ method, these parameters are used to set the object's first_name, last_name, and course_name attributes, through the parent

Person class (for first_name and last_name). Because this was established in the parent class (Person).

After the __init__ method completes, the new Student object is fully initialized and the variable student holds a reference to this object.

Now *we can use the* ***'student' variable to access the attributes and methods defined in the Student*** (and Person) class. For example, we can print the student's full name and enrolled course by calling **print(student)**, which would use the **__str__ method defined in the Student class to return a formatted string**.

```
    @staticmethod
    def input_student_data():
        """ Gets student data from the user and returns a Student object

        ChangeLog: (Who, When, What)
        Nelly,22.11.2023,Created function
        :param None
        :return: Student object
        """

        try:
            student_first_name = input("Enter the student's first name: ").strip()
            student_last_name = input("Enter the student's last name: ").strip()
            course_name = input("Please enter the name of the course: ").strip()
            student = Student(student_first_name, student_last_name, course_name)
            print()
            print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
            return student
        except ValueError as e:
            IO.output_error_messages(message="One of the values was the correct type of data!", error=e)
        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
```

Figure 7. Input Student_data

Summary

The objective of this assignment was as in the previous assignments, to 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 Assignment07, but this assignment demonstrates how to create and use classes to manage data and with structured error handling

This document includes a description of the steps that I followed while learning about objects, classes, inheritance and Python magic methods such as __str__() "to string" method.

Something that was for me new to learn is the use of objects or instances which particularly defines Python that is an object oriented language.

Citations

1. Writing professional papers:
https://www.youtube.com/watch?v=9ojhSW9ljjo&feature=youtu.be

2. Open AI ChatGPT, Oct. 2023, chat.openai.com/chat: A few aspects of this assignment were informed by queries submitted to the ChatGPT.

3. Slides and videos from class, laboratories and Demo/Videos of the course in this Module 7.

4. Starting script included in the Module 07 materials from our class. I used this script to start this Assignment 07.