Andrew Hopkins
February 17, 2024
IT FDN 110 A
Assignment 07
https://github.com

Classes and Objects

Introduction

This week's assignment covered Classes and Objects but more specifically Data Classes, Data Class Components, Constructors, and Properties. There were a lot of ups and downs for me while I was writing this assignment. All of the concepts that were covered came easy and made sense to me. Establishing and understanding what data classes were, how they functioned internally, and the point of the constructors and properties was all very clear. The part where I struggled the most actually was keeping track of all of the new references and how to adjust the initial code to accommodate the use of the new data classes.

Writing the Script.

Creating the "Person" Data Class

I knew a couple of facts before I started on the Person class: I was going to have to establish first and last name constructors and properties to be called upon by another data class and because these properties were going to be called upon I would have to handle exceptions with them as well.

The first thing I did was define the __initi__ method with attributes I knew would be passed to it and created private constructors to be equal to those attributes. I then created an individual getter and setter property for both first name and last name, ensuring that I was handling any user input errors on the setters (in this case, making sure they were alphabetic inputs). The last thing I did was overwrite the string method of the class to display a custom output of the constructors

```python
class Person:
    """..."""

    # TODO Add first_name and last_name properties to the constructor (Done)
    def __init__(self, first_name: str = "", last_name: str = ""):
        self.__first_name = first_name
        self.__last_name = last_name

    # TODO Create a getter and setter for the first_name property (Done)
    @property
    def first_name(self):
        return self.__first_name

    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha():
            self.__first_name = value
        else:
            raise ValueError("Please enter a first name that only contains letters")

    # TODO Create a getter and setter for the last_name property (Done)
    @property
    def last_name(self):
        return self.__last_name

    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha():
            self.__last_name = value
        else:
            raise ValueError("Please enter a last name that only contains letters")

    # TODO Override the __str__() method to return Person data (Done)
    def __str__(self):
        return f'{self.__first_name}, {self.__last_name}'
```

Figure 1: Setting up the properties and error handling for the Person Class

Creating the "Student" Data Class

I knew creating the Student Class was going to require a little bit different of a setup because it needed to inherit properties from the Person Class. The first step was to just define the class and its attributes. But because it had to inherit properties from the Person Class the next immediate step was to call on the super().__init__ method to dictate that the class should set the first name

and last name to those established by the Person Class. A Student did require the added constructor of having a course name associated with it so I defined that constructor, created getter and setter properties for it with error handling for user input (since this input is alphanumeric the only exception I wanted it to handle was if for some reason the user didn't input anything). Finally, I did the same as with the Person Class and overwrote the string method to display a custom output of the object.

```
# TODO Create a Student class the inherits from the Person class (Done)
class Student(Person):
    """..."""

    # TODO call to the Person constructor and pass it the first_name and last_name data (Done)
    def __init__(self, student_first_name: str = "", student_last_name: str = "", course_name: str = ""):
        super().__init__(first_name=student_first_name, last_name=student_last_name)
        # Properties for student first name and last name are inherited from Person class

        # TODO add a assignment to the course_name property using the course_name parameter (Done)
        self.__course_name = course_name

    # TODO add the getter for course_name (Done)
    @property
    def course_name(self):
        return self.__course_name

    # TODO add the setter for course_name (Done)
    @course_name.setter
    def course_name(self, value: str):
        if value != '':
            self.__course_name = value
        else:
            raise ValueError('Please enter a course name')

    # TODO Override the __str__() method to return the Student data (Done)
    def __str__(self):
        return f'{self.first_name}, {self.last_name}, {self.course_name}'
```

Figure 2: Setting up the properties and error handling for the Student Class

Adjusting the Read and Write to File Functions

The rest of the code, from the previous assignments was set to read and write to a file using dictionary data, however that wasn't going to do me any good when I needed to call on the Student class. Because of this I had to convert the dictionary style data in the JSON file into a list of data class components and vice versa. In order to read in the data I created a list of the JSON entries then iterated over that list while defining each field as its corresponding property. Then, in the other direction, I had to convert a list of Student class properties into a dictionary to properly write it to the file. This was more or less the opposite process. I created a list to put the dictionary entries into, then iterated over a list of Student Class data and assigned each property

to the corresponding dictionary field. Once that was done it was just a matter of dumping the new list into the JSON file.

```
def read_data_from_file(file_name: str, student_data: list):
    """This function reads data from a json file and loads it into a list of dictionary rows..."""

    try:
        file = open(file_name, "r")
        list_of_dict_data = json.load(file)
        for student in list_of_dict_data:
            student_object: Student = Student(student_first_name=student["FirstName"],
                                              student_last_name=student["LastName"],
                                              course_name=student["CourseName"])
            student_data.append(student_object)
        file.close()
    except Exception as e:
        IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)

    finally:...
    return student_data

@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    """...."""

    try:
        list_of_dict_data: list = []
        for student in student_data:
            student_json: dict \
                = {"FirstName": student.first_name, "LastName": student.last_name,
                   "CourseName": student.course_name}
            list_of_dict_data.append(student_json)
        file = open(file_name, "w")
        json.dump(list_of_dict_data, file)
        file.close()
        IO.output_student_and_course_names(student_data=student_data)
    except Exception as e:...
    finally:...
```

Figure 3: Converting between JSON Dictionary data and Lists

Adjusting the Input and Output Functions

In the initial version of this code the input and output functions were calling on specific variables and handling exceptions for each of them depending on the input. Since the Student Class data had taken the place of those variables and was also handling exceptions I was able to trim down these functions by simply calling on the Student class for the information I needed. The output was straightforward as all it required was to iterate over a list of Student Class data and output an f string containing calls to the first name, last name, and course name properties. The adjustment to the input was similar. I eliminated the error handling aspects of the previous code and instead of calling on specific variables I set Student Class data and properties to be equal to

the input from the user. Once that was completed the list of Student Class data was amended to include the new user input.

```
def output_student_and_course_names(student_data: list):
    """This function displays the student and course names to the user..."""

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

@staticmethod
def input_student_data(student_data: list):
    """This function gets the student's first name and last name, with a course name from the user..."""

    try:
        student = Student()
        student.first_name = input("Enter the student's first name: ")
        student.last_name = input("Enter the student's last name: ")
        student.course_name = input("Enter the name of the course: ")
        student_data.append(student)
        print()
        print(f"You have registered {student.first_name} {student.last_name} for {student.course_name}.")
    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)
    return student_data
```

Figure 4: Calling on the Student Class for input and output

Notes

- The way that the PyCharm debugger represents objects like Student initially made me believe that I was reading in data incorrectly because I expected it to be formatted more like a list or a dictionary and it wasn't until I read through it very carefully that it has its own syntax.

Summary

I can sit here now and easily explain how the additions of the Data Classes worked and now function with the rest of the code but the truth is that it took me a long time to arrive at this point. The concepts of the Data Classes, their properties, inheritance, etc. all, again, made a lot of sense as I was learning about and practicing with them. It was the addition of a new potential reference and having to adjust for that that really started making things difficult for me. As I was adjusting the code much of the data that I was passing became blurred for me. I don't think the naming convention helped but I was starting to get confused about whether I was passing a dictionary, or a list, or a list of dictionaries or, an object, and how was I supposed to iterate of those again? It genuinely required a lot of careful debugging to finally see what was happening and get the final product correct for what was the expected functionality of this assignment. All of that having been said, I look forward to using the inheritance aspect even more as I can see the huge potential it has across data sets that have similar attributes.