Name: Mike Luke

Date: 11/24/2025

Course: IT FDN 110 A Au 25: Foundations Of Python Programming

Assignment 07 Code Repository

Assignment 07 –

Introduction

For Week 7, we are introduced to constructors, attributes, inheritance, and the overridden method. Week 6 was great with helping to organize code and set structure, but for this week we learned how to set a parent class that can be used to inherit attributes into a child class. This can be super helpful when there is commonality in your data. It allows you to build a base and add additional data depending on the requirement.

Topic

I simply started with the Assignment07-Starter.py file and worked my way through the "TODO" list.

```
# TODO Create a Person Class
# TODO Add first_name and last_name properties to the constructor
# TODO Create a getter and setter for the first_name property
# TODO Create a getter and setter for the last_name property
# TODO Override the __str__() method to return Person data

# TODO Create a Student class the inherits from the Person class
# TODO call to the Person constructor and pass it the first_name and last_name data
# TODO add a assignment to the course_name property using the course_name parameter
# TODO add the getter for course_name
# TODO add the setter for course_name
# TODO Override the __str__() method to return the Student data
```

Setting the class for Person was similar to the previous week, but instead we set up and initialize its attributes. Using the getter makes it readable and the setter controls what happens when you assign the values.

```python
# TODO Create a Person Class
class Person:

# TODO Add first_name and last_name properties to the constructor
    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
    @property
    def first_name(self):
        return self.__first_name.title()
    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha() or value == '':
            self.__first_name = value
        else:
            raise ValueError('The first name must not contain numbers .')
# TODO Create a getter and setter for the last_name property
    @property
    def last_name(self):
        return self.__last_name.title()
    @first_name.setter
    def last_name(self, value: str):
        if value.isalpha() or value == '':
            self.__last_name = value
        else:
            raise ValueError('The first name must not contain numbers .')
```

Setting up the Student class allows us to set a child class to Person. This allows us to pull the first_name and last_name and add additional data like course_name. This is useful if we were to have a list of all students and staff at a school. The commonality is the first and last names, but we can then modify this data to meet our requirements of what we would want to see for a student opposed to a teacher. Students will be registered for course, but a teacher would be assigned a course to instruct or multiple courses to instruct. Also by using the setter we are able to add in error handling allowing us to remove some of the previous code we used for week 6. It helps to simplify and make the code more modular.

```python
# TODO Override the __str__() method to return Person data
    def __str__(self):
        return f'{self.first_name} {self.last_name}'
# TODO Create a Student class the inherits from the Person class
class Student(Person):
    def __init__(self, first_name: str = '', last_name: str = ''):
        super().__init__(first_name, last_name)
        self.course_name = course_name

        @property
        def course_name(self):
            return self.__course_name.title()

        @course_name.setter
        def course_name(self, value: str):
            try:
                self.__course_name = str(value)
            except ValueError as e:
                raise ValueError('Invalid course name.')
```

The override functionality allows us to control the output or turn it into a format we would like.


```python
# TODO Override the __str__() method to return Person data
    def __str__(self):
        return f'{self.first_name} {self.last_name}'
# TODO Create a Student class the inherits from the Person class
class Student(Person):
    def __init__(self, first_name: str = '', last_name: str = ''):
        super().__init__(first_name, last_name)
        self.course_name = course_name

        @property
        def course_name(self):
            return self.__course_name.title()

        @course_name.setter
        def course_name(self, value: str):
            try:
                self.__course_name = str(value)
            except ValueError as e:
                raise ValueError('Invalid course name.')
```


The next obstacle was taking the json data and creating a dictionary from the data and then appending to our student_data table. In order to write the data to a file we were instructed to create a dictionary then take the data and dump it into our file.

```python
class FileProcessor:
    def read_data_from_file(file_name: str, student_data: list):
        """
        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        RRoot,1.5.2030,Converted list of dictionaries to list of student objects

        :param file_name: string data with name of file to read from
        :param student_data: list of dictionary rows to be filled with file data

        :return: list
        """
        try:
            file = open(file_name, "r")

            list_of_dictionary_data = json.load(file)  # the load function returns a list of dictionary r
            for student in list_of_dictionary_data:  # Convert the list of dictionary rows into Student o
                student_object: Student = Student(first_name=student["FirstName"],
                                                  last_name= student["LastName"],
                                                  course_name=student["CourseName"])
                student_data.append(student_object)

            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages( message: "Text file must exist before running this script!", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

```python
class FileProcessor:
    def write_data_to_file(file_name: str, student_data: list):
        """ This function writes data to a json file with data from a list of dictionary rows

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        RRoot,1.2.2030,Converted code to use student objects instead of dictionaries

        :param file_name: string data with name of file to write to
        :param student_data: list of dictionary rows to be writen to the file

        :return: None
        """
        try:
            list_of_dictionary_data: list = []
            for student in student_data:  # Convert List of Student objects to list of dictionary rows.
                student_json: dict \
                    = {"FirstName": student.first_name, "LastName": student.last_name, "CourseName": student.course_name}
                list_of_dictionary_data.append(student_json)

            file = open(file_name, "w")
            json.dump(list_of_dictionary_data, file)
            file.close()
        except TypeError as e:
            IO.output_error_messages( message: "Please check that the data is a valid JSON format", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
```

For Menu 2, we change it to print our objects.

```python
class FileProcessor:
    def read_data_from_file(file_name: str, student_data: list):
        """
        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        RRoot,1.5.2030,Converted list of dictionaries to list of student objects

        :param file_name: string data with name of file to read from
        :param student_data: list of dictionary rows to be filled with file data

        :return: list
        """
        try:
            file = open(file_name, "r")

            list_of_dictionary_data = json.load(file)  # the load function returns a list of dictionary r
            for student in list_of_dictionary_data:  # Convert the list of dictionary rows into Student o
                student_object: Student = Student(first_name=student["FirstName"],
                                                  last_name= student["LastName"],
                                                  course_name=student["CourseName"])
                student_data.append(student_object)

            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages( message: "Text file must exist before running this script!", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

```python
class FileProcessor:
    def write_data_to_file(file_name: str, student_data: list):
        """ This function writes data to a json file with data from a list of dictionary rows

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        RRoot,1.2.2030,Converted code to use student objects instead of dictionaries

        :param file_name: string data with name of file to write to
        :param student_data: list of dictionary rows to be writen to the file

        :return: None
        """
        try:
            list_of_dictionary_data: list = []
            for student in student_data:  # Convert List of Student objects to list of dictionary rows.
                student_json: dict \
                    = {"FirstName": student.first_name, "LastName": student.last_name, "CourseName": student.course_name}
                list_of_dictionary_data.append(student_json)

            file = open(file_name, "w")
            json.dump(list_of_dictionary_data, file)
            file.close()
        except TypeError as e:
            IO.output_error_messages( message: "Please check that the data is a valid JSON format", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
```

Similarly, we capture the input data from option 1 in a student variable that is defined as Student() and append that to our table.

```python
class FileProcessor:
    def read_data_from_file(file_name: str, student_data: list):
        """
        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        RRoot,1.5.2030,Converted list of dictionaries to list of student objects

        :param file_name: string data with name of file to read from
        :param student_data: list of dictionary rows to be filled with file data

        :return: list
        """
        try:
            file = open(file_name, "r")

            list_of_dictionary_data = json.load(file)  # the load function returns a list of dictionary r
            for student in list_of_dictionary_data:  # Convert the list of dictionary rows into Student o
                student_object: Student = Student(first_name=student["FirstName"],
                                                  last_name= student["LastName"],
                                                  course_name=student["CourseName"])
                student_data.append(student_object)

            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages( message: "Text file must exist before running this script!", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

```python
class FileProcessor:
    def write_data_to_file(file_name: str, student_data: list):
        """ This function writes data to a json file with data from a list of dictionary rows

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        RRoot,1.2.2030,Converted code to use student objects instead of dictionaries

        :param file_name: string data with name of file to write to
        :param student_data: list of dictionary rows to be writen to the file

        :return: None
        """
        try:
            list_of_dictionary_data: list = []
            for student in student_data:  # Convert List of Student objects to list of dictionary rows.
                student_json: dict \
                    = {"FirstName": student.first_name, "LastName": student.last_name, "CourseName": student.course_name}
                list_of_dictionary_data.append(student_json)

            file = open(file_name, "w")
            json.dump(list_of_dictionary_data, file)
            file.close()
        except TypeError as e:
            IO.output_error_messages( message: "Please check that the data is a valid JSON format", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
```

Summary

The assignment this week was difficult to understand, and I found that the readings, videos, and additional resources were missing some details for the assignment. This is my first programming class, so it took quite a bit of time and effort to work through all the problems I ran into. It did help to build on what we have learned previously and introduced the parent and child classes, which I can imagine will be useful in the future. It will allow us to take in data that is shared, create modular code that can be called to retrieve/process data, and add additional data if required.