Jake Jake Velasco

Feb. 28th, 2024

IT FDN 110 A Wi 24

Assignment 07

Objects & Classes

Introduction

In this assignment the goal is to learn about how to create and use classes for management data purposes. Similar to the previous assignment, I will modify existing code and demonstrate my knowledge of creating sets of data classes, error handling, data validation, and learning how to directly share and/or privatize my code via PyCharm to GitHub.

Create Person Class Properties Constructor Method

Mod07-Lab01:Working with Constructors as well as the examples within the Mod07-Notes document were heavily leveraged to create my person class, add the first_name and last_name aka properties to the constructor using the self method.

```
import json

# Define the Data Constants
MENU: str = '''
---- 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.
-----------------------------------------
'''
FILE_NAME: str = "Enrollments.json"

# Define the Data Variables
students: list = []  # a table of student data
menu_choice: str  # Hold the choice made by the user.

# TODO Create a Person Class
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:
    JJVelasco, 2.28.2024: Created the class.
    """
# 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
```

Getter (Properties), Setter for First & Last Name within Person Class

Looking back on the recorded lecture guided me through to setting up the property aka getter and setter attributes for my “variables” first & last for the student. I embedded human error handing by utilizing the if statements to put alphabetic characters only and the value error function to spit out a message if this was not followed suit.

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

    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha():
            self._first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")

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

    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha() or value == "":
            self._last_name = value
        else:
            raise ValueError('The last name should not contain numbers.')
```

Person Class Last Step Return the String

To ensure that the result of the new code is consistent as the, I added data validation to account for when the parameters use the default empty string, so that I can create a student object without arguments and can assign the properties values afterward. I arrived at the following with minimal debugging.

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

Passing Person into the Student Class aka Inheritance

Next segment became natural and the use of when adding a class name to indicate explicit inheritance between the Person and Student class. The student class gained all the attributes and methods from the person class, which allowed me to reuse and extend the functionally defined the in the person class with wasting time and duplicating code.

```
# TODO Create a Student class the inherits from the Person class (Done)
class Student(Person):
    """
        A collection data about students

        ChangeLog: (Who, When, What)
        JJVelasco, 2.28.2024, Created class and added properties and private
attributes
        """
# 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)

# 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):
        self._course_name = value

# 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}'
```

Processing: Modify Class FileProcessor segment

To ensure the converted code work with the json file to use student object instead of dictionaries I arrived at the following code. I leveraged the Mod07-Lab02: Working with Class Properties examples. The most difficult part was figuring out to modify the json read and write methods.

```
# Processing ------------------------------------- #
class FileProcessor:
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
```

```
    RRoot,1.1.2030,Created Class
    JJVelasco, 2.28.2024, Converted code to use student objects instead of
dictionaries
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list[Student]) ->
list[Student]:
        """ This function reads data from a json file and loads it into a
list of dictionary rows

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        JJVelasco, 2.28.2024, Converted code to use student objects instead
of dictionaries

        :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)
           for student in list_of_dictionary_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("There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list[Student]):
        """ 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
        JJVelasco, 2.28.2024, 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:
                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 Exception as e:
            IO.output_error_messages(message="Error: There was a problem with
reading the file.", error=e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

Presentation Reformatting & Adjusting the I/O Functions

Modifications were made to replace the previous variables and exception handling within the previous input and output functions to cater to the student class data which has replaces variables. Much was leaned out. The key was to ensure that the previous variables within the f string in relation to the first/last/course name properties matched the student data class properties.

```
# Presentation ------------------------------------ #
class IO:
    """
    A collection of presentation layer functions that manage user input and
output

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    RRoot,1.2.2030,Added menu output and input functions
    RRoot,1.3.2030,Added a function to display the data
    RRoot,1.4.2030,Added a function to display custom error messages
    JJVelasco, 2.28.2024, Converted code to use student objects instead of
dictionaries
    """

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """ This function displays the a custom error messages to the user

        ChangeLog: (Who, When, What)
        RRoot,1.3.2030,Created function

        :param message: string with message data to display
        :param error: Exception object with technical message to display

        :return: None
        """
        print(message, end="\n\n")
```

```
        if error is not None:
            print("-- Technical Error Message -- ")
            print(error, error.__doc__, type(error), sep='\n')

    @staticmethod
    def output_menu(menu: str):
        """ This function displays the menu of choices to the user

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function


        :return: None
        """
        print()  # Adding extra space to make it look nicer.
        print(menu)
        print()  # Adding extra space to make it look nicer.

    @staticmethod
    def input_menu_choice():
        """ This function gets a menu choice from the user

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function

        :return: string with the users choice
        """
        choice = "0"
        try:
            choice = input("Enter your menu choice number: ")
            if choice not in ("1","2","3","4"):  # Note these are strings
                raise Exception("Please, choose only 1, 2, 3, or 4")
        except Exception as e:
            IO.output_error_messages(e.__str__())  # Not passing e to avoid
the technical message

        return choice

    @staticmethod
    def output_student_and_course_names(student_data: list[Student]):
        """ This function displays the student and course names to the user

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        JJVelasco, 2.28.2023, Changed to use student objects instead of
dictionaries

        :return: None
        """

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

    @staticmethod
```

```
    def input_student_data(student_data: list[Student]) -> list[Student]:
        """ This function gets the student's first name and last name, with a
course name from the user

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        JJVelasco, 2.28.2024, Relocated error handling for names to class
function

        :param student_data: list of dictionary rows to be filled with input
data

        :return: list
        """

        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("Please 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


# Start of main body

# When the program starts, read the file data into a list of lists (table)
# Extract the data from the file
students = FileProcessor.read_data_from_file(file_name=FILE_NAME,
student_data=students)

# Present and Process the data
while (True):

    # Present the menu of choices
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    # Input user data
    if menu_choice == "1":  # This will not work if it is an integer!
        students = IO.input_student_data(student_data=students)
        continue

    # Present the current data
    elif menu_choice == "2":
        IO.output_student_and_course_names(student_data=students)
```

```
        continue

    # Save the data to a file
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(file_name=FILE_NAME,
student_data=students)
        continue

    # Stop the loop
    elif menu_choice == "4":
        break  # out of the loop
    else:
        print("Please only choose option 1, 2, or 3")

print("Program Ended")
```

Summary

Even after rewatching the demo/videos, and recorded lecture, and reviewing the assignment video this concept is still foreign to me. However, by thinking of real time examples of how this can organize data capturing within my workspace/company, this can be extremely useful in providing quality data clean up.