Ryan Seng

5/25/2024

Python 100

Assignment 07

Assignment 07 – Classes and Objects

Introduction

This document will outline how I created the assignment06.py script to turn in for the assignment. To create this code, I referenced the lessons learned in the class on 5/22/2024 and the Lab answer script for Lab 4 – Using Inheritance. Lastly, the code was built off of Assignment06.py. The methodology section below goes into further detail.

Methodology

The comments for the assignment were taken straight from my submission for Assignment 06 but I changed <Your Name Here>, <Date>, and Assignment to my name, Ryan Seng, 5/25/2024, and Assignment07 which was when I initially started working on the assignment. The snip below contains the text I was referring to:

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment07
# Desc: This assignment demonstrates using lists and files to work with data
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Ryan Seng, 5/25/2024, Assignment07
# ------------------------------------------------------------------------------------------ #
```

Since the assignment required that the “Enrollments” file I worked with be a .json file, I imported the functions from the json library using “import json”

For the constants and variables needed outside of each class or function, there is only 2 constants and 2 variables. The first constant is MENU which is a string with text resembling a menu for users to give inputs for. The second constant is FILE_NAME which is also a string set equal to “Enrollments.json”. For the variables, there is menu_choice which is set as a blank string and students which is a blank list. Below is a snip with the code as described:

```
# Importing JSON library to get json functions
import json

# This is the MENU users will be selecting from
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
----------------------------------------
"""

# This are the other constants
FILE_NAME: str = "Enrollments.json"

# These are the variables
menu_choice: str = ""
students: list = []
```

For the next part of the assignment, I created 4 classes which would contain functions that can be used throughout the script. The first class is the FileProcessor class which contains two functions with static method decorators and was taken from Assignment06.py. The first function is the read_data_from_file function which takes a file_name and student_data as arguments. It starts with opening the file in file_name in read mode using "with open()". The script then read the data from the file into the loaded_student_data variable using json.load(). Then it iterated through each dictionary entry on the file, ran the entry through the Student class to create a Student Object (more on this later), and appended the object to the student_data list variable. Then it would return the student_data variable. There are some exceptions in case the file name could not be found, if there were any issues with the file itself or if there was some error I did not name directly. These exceptions would then cause another function in another class to be run. Lastly, to be safe, I included a clause in the code from the Assignment06-Starter.py to ensure the file was closed in case the file was somehow open. Below is the code as described.

```
class FileProcessor:
    """
    This function reads data in from the file in file_name and puts it into a list.

    Change Log: (Who, When, What)
        Ryan Seng, 5/19/2024, Created Assignment06
        Ryan Seng, 5/25/2024, Updated Script for Assignment07 version update
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        try:
            with open(file_name, 'r') as file:

                loaded_student_data = json.load(file)
                for student in loaded_student_data:
                    loaded_student = Student(first_name = student["First_Name"], last_name = student["Last_Name"], course = student["Course"])
                    student_data.append(loaded_student)

                print("File loaded")
        except FileNotFoundError as e:
            IO.output_error_messages("The text/json file could not be found when running this script", e)
        except ValueError as e:
            IO.output_error_messages("There are corruption issues in the file", e)
        except Exception as e:
            IO.output_error_messages("Unknown error.", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

The other function in the FileProcessor class is write_data_to_file. This function would take a file_name and student_data passed in as arguments. Like the "read" function in FileProcessor, the script starts with an empty list called list_of_students. Then it iterates through each object passed in the student_data list of objects, and puts each object's first_name, last_name, and course as one dictionary entry. Then the script would append that dictionary entry into the list_of_students. This portion was taken from the lessons learned and Lab 4's answer script. For the last portion, it would try to open the file named in file_name, write the list_of_students into that file, print the that the student list was saved into the file, then close the file. If there were any errors with the data type or an error I didn't anticipate, the code would run IO.output_error_messages(). Lastly, like with the previous group of code, I included code from Assignment06-Starter.py that covered the case that if the file was still open, it would close. Below is the code as described.

```
"""
This function takes a file name and a list of student data and writes the student data into a file named in file_name

Change Log: (Who, When, What)
    Ryan Seng, 5/19/2024, Created Assignment06
    Ryan Seng, 5/25/2024, Updated Script for Assignment07 version update
"""
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    try:
        list_of_students: list = []
        for student in student_data:
            student_json: dict = {"First_Name": student.first_name, "Last_Name": student.last_name, "Course": student.course}
            list_of_students.append(student_json)

        file = open(file_name, "w")
        json.dump(list_of_students, file)
        print(f"The student list was saved in {FILE_NAME}")
        file.close()
    except TypeError as e:
        IO.output_error_messages("The data may not be in a valid format", e)
    except Exception as e:
        IO.output_error_messages("Unknown error.", e)
    finally:
        if file.closed == False:
            file.close()
```

The next chunk of code for the assignment pertains to the IO class and its functions. All functions have the static method decorator. The functions output_error_messages, output_menu and input_menu_choice were not changed from Assignment06.py and only changelogs were added. The first function is the output_error_messages function which takes in a message as a string, and an error. If no error is passed into the function, then the default is None. The function would print the message, then if the error is not None, it'd print details for the error. The "if" portion of the code was taken from Lab 3. Below is the code as described.

```
class IO:
    """
    This function prints error messages out in a way that should be legible and understandable to
    anyone working on this script.

    Change Log: (Who, When, What)
        Ryan Seng, 5/19/2024, Created Assignment06
    """
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        print(message)
        if error is not None:
            print("-- Technical Error Message --")
            print(error, error.__doc__, type(error), sep='\n')
```

For the second function, it is named output_menu which takes a menu variable as an argument and prints that menu.

For the third function, it is called input_menu_choice. The function prompts the user to select an option by giving an input. If the input is no "1", "2", "3", or "4", then it would raise an error. Additionally, if there was some error that the user entered that I did not anticipate, then it would also raise an error and call the output_error_messages function in the IO class. Lastly, it would return the user's input. Below is the code for both the second and third functions as described.

```
    """
    This function prints out a set of text that usually should be a menu that gives users an idea
    of what to input for the input_menu_choice() function

    Change Log: (Who, When, What)
        Ryan Seng, 5/19/2024, Created Assignment06
    """
    @staticmethod
    def output_menu(menu: str):
        print(menu)

    """
    This function prompts the user to select an option. If they don't select 1, 2, 3, or 4, then
    it prompts them to try to select one of those options again

    Change Log: (Who, When, What)
        Ryan Seng, 5/19/2024, Created Assignment06
    """
    @staticmethod
    def input_menu_choice():
        try:
            user_choice = input("Please select an option: ")
            if user_choice not in ("1","2","3","4"):
                raise Exception("Please select only 1, 2, 3, or 4.")
        except Exception as e:
            IO.output_error_messages("Unknown Error.",e.__str__())
        return user_choice
```

The fourth function is the output_student_courses function which takes in a list of student_data as the argument. The function would then print the message “The current list of students is:” and the list of student_data passed in. The code was changed to print the objects in the list (as seen in the picture below)

```
    """
    This function prints the current list of student data (including inputs from the user)

    Change Log: (Who, When, What)
        Ryan Seng, 5/19/2024, Created Assignment06
        Ryan Seng, 5/25/2024, Updated Script for Assignment07 version update
    """
    @staticmethod
    def output_student_courses(student_data: list):
        print("The current list of students is:")
        print("Name \t\tLast Name \tCourse")
        for student in student_data:
            print(f"{student.first_name} \t\t {student.last_name} \t\t{student.course}")
```

The final function in the IO class is input_student_data with a static method decorator and it takes a list of student_data as an argument. The code would prompt the user to try to enter a student’s first name, last name, and course. Then it would run the first_name, last_name, and course into the Student class, create an Student object called new_student, and append it to the student_data list. Then the function would return the student_list variable. If there were symbols or numbers entered in the student’s name, then it would raise an error. This error handling was taken from Lab 3 as I did not know the .isalpha() function existed. Below is the code as described.

```
    """
    This function takes in users' inputs for a student's first name, last name, and course, create a student object using those inputs,
    and adds that student to the list of student data

    Change Log: (Who, When, What)
        Ryan Seng, 5/19/2024, Created Assignment06
        Ryan Seng, 5/25/2024, Updated Script for Assignment07 version update
    """
    @staticmethod
    def input_student_data(student_data: list):
        try:
            student_first_name = input("Please enter the student's first name: ")
            if not student_first_name.isalpha():
                raise ValueError("The first name should not contain numbers or symbols")

            student_last_name = input("Please enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError("The first name and last name should not contain numbers or symbols")

            course_name = input("Please enter the course name: ")

            new_student = Student(first_name=student_first_name, last_name=student_last_name, course= course_name)

            student_data.append(new_student)
            print(f"{new_student.first_name} {new_student.last_name} in {new_student.course} has been added.")
        except ValueError as e:
            IO.output_error_messages("There is a incorrect value.", e)
        except Exception as e:
            IO.output_error_messages("Unknown error.", e)
        return student_data
```

As part of Assignment 07, 2 classes had to be created. The first is the Person class which is an object with a first_name, and a last_name as attributes/properties. When a first_name and last_name are passed in as arguments, it sets those as its properties and returns the object. The second is the Student class which is a child of the Person class. It uses the same baseline coding as Person but takes in a course argument and sets that argument as its property. Then it returns the object. These two classes facilitate the usage of objects throughout the code and can be seen below.

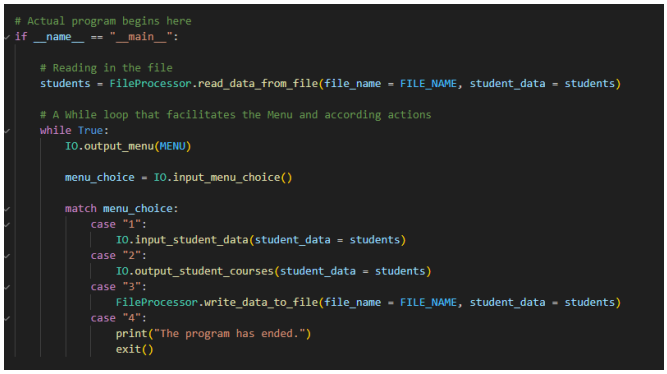
```
# This class is the parent object to the student class and is the baseline for objects in this program
class Person:

    def __init__(self, first_name: str = "", last_name: str = ""):
        self.first_name = first_name
        self.last_name = last_name

    @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("There are numbers or symbols in the first name!")

    @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("There are numbers or symbols in the first name!")

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

```
# This is a child class of the Person class. It builds off of the Person class
# by taking courses that the person is taking
class Student(Person):

    def __init__(self, first_name: str, last_name: str, course: str):
        super().__init__(first_name = first_name, last_name= last_name)
        self.course = course

    @property
    def course(self):
        return self.__course

    @course.setter
    def course(self, value: str):
        self.__course = value

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

The last remainder of the code is the main program itself which hasn't changed from Assignment06.py. The script is only set to run if the script is the main program due to the "if __name__ == "__main__"". If the script is the main program, then it will load the student data using the FileProcessor class' read_data_from_file function with FILE_NAME and students as the arguments. Then it stores the returned data in the students variable. Then the script prints the menu using the IO.output_menu with the MENU constant as the argument. Then it runs IO.input_menu_choice() and stores the result in the menu_choice variable.

If the menu_choice variable is 1, then the script runs the IO.input_student_data function with the students variable as the argument. If menu_choice is 2, then it runs the IO.output_student_courses

function with students as the argument. If the menu_choice is 3, then it runs the FileProcessor.write_data_to_file function with FILE_NAME and students as the arguments. Lastly, if menu_choice is 4, then it prints “the program has ended” and exits the program. Below is the code as described in the last 2 paragraphs.



```
# Actual program begins here
if __name__ == "__main__":

    # Reading in the file
    students = FileProcessor.read_data_from_file(file_name = FILE_NAME, student_data = students)

    # A While loop that facilitates the Menu and according actions
    while True:
        IO.output_menu(MENU)

        menu_choice = IO.input_menu_choice()

        match menu_choice:
            case "1":
                IO.input_student_data(student_data = students)
            case "2":
                IO.output_student_courses(student_data = students)
            case "3":
                FileProcessor.write_data_to_file(file_name = FILE_NAME, student_data = students)
            case "4":
                print("The program has ended.")
                exit()
```

By doing all of this, I was able to complete Assignment 07. I then named the script “assignment07.py”.

Summary

Through lessons learned in class and from Lab 4, I was able load in data from a csv file into a list to create a list of objects. Then I prompted the user to follow a menu with a variety of options including registering data as objects, displaying the current data, saving the data, and exiting the program. All of this included structured error handling detailing what went wrong if there was an input from Enrollment.csv or from the user that cause issues.

Link to GitHub

Link to GitHub: https://github.com/RyanS39/IntroToProg-Python-Mod07