

Jacob Widner

8/11/2024

IT FDN 110 A Su 24: Foundations of Programming: Python

Assignment07

GitHub link: <https://github.com/JakeWidner/IntroToProg-Python-Mod07>

Using Class Objects for Data

Introduction

In this document I will explain the process I used to complete the seventh assignment for this course. The script used to complete this assignment builds on what was used for the previous assignments and adds new concepts such as using class objects to store and manipulate data.

Designing the Code

Like the previous assignments, this script asks the user for input. This assignment asks the user to choose an option from a menu. The options include opportunities for user input, printing output to the screen, saving data to a file, and exiting the program. This assignment also reads data from a JSON file that was previously saved to append the data. An important difference between this assignment and the last assignment is that the data read from the JSON file is converted into a list of objects and manipulated in that state.

The Script Header

In the instructional materials for the first module, we were introduced to the concept of the script header. The script header is usually placed at the top or “head” of the script and consists of a series of commented lines that hold useful information about the script. These elements include a title, description, and a change log.¹ The header for this assignment (Figure 1) contains the same basic information used in the previous assignments.

```
# ----- #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
#   JWidner, 8/11/2024, Created Script
#   <Your Name Here>, <Date>, <Activity>
# ----- #
```

Figure 1: Assignment07 Script Header.

¹ Root, Randal, Creating Python Scripts: Script Headers, *Mod01-Notes*, p17.

Importing JSON Library

This assignment requires the manipulation of JSON files.² To use JSON files in your script you must first import the module that contains the appropriate methods for manipulating JSON files³ (Figure 2).

```
# import json module
import json
```

Figure 2: JSON Module Import

Defining the Constants

We were introduced to the concepts of variables, constants and data types⁴ as well as type hints⁵ in the first module. For this assignment, a set of constants were defined including the menu that will be displayed to the user and the name of the JSON file we will be using to write the data to (Figure 3).

```
# 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"
```

Figure 3: Assignment07 Constants.

Defining the Variables

For this assignment, only two variables were used. One variable was defined as an empty string⁶ and one was defined as an empty list⁷. Additional variables would be defined and used inside functions defined later in the code. These variables would be filled with values later in the code (Figure 4).

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

Figure 4: Assignment07 Variables.

Class and Function Definitions

This assignment calls for the functional code to be organized into two classes⁸ with descriptive document strings⁹ and seven functions¹⁰. The functions within these two classes use the “@staticmethod” modifier to allow the class functions to be used directly.¹¹ Furthermore, two additional object classes¹² are defined

² Root, Randal. JSON Files, *Mod05-Notes*, p13.

³ Root, Randal. JSON Files: Working with JSON Files, *Mod05-Notes*, p15.

⁴ Root, Randal, Program Data: Constants, Variables and Data Types, *Mod01-Notes*, p27.

⁵ Root, Randal, Data Types: Data Type Naming Conventions, *Mod01-Notes*, p33.

⁶ Root, Randal. Common Types of Errors, *Mod02-Notes*, p28.

⁷ Root, Randal. Data Collections: Lists, *Mod04-Notes*, p17.

⁸ Root, Randal. Classes and Functions, *Mod06-Notes*, p27.

⁹ Root, Randal. Classes and Functions: Document Strings (Docstrings), *Mod06-Notes*, p30.

¹⁰ Root, Randal. Organizing Your Code: Functions, *Mod06-Notes*, p2.

¹¹ Root, Randal. Classes and Functions: Static Classes, *Mod06-Notes*, p29.

¹² Root, Randal. Statements, Functions and Classes: Objects vs. Classes, *Mod07-Notes*, p4.

to create objects to hold the student data. The following sections will describe the class objects and functional classes.

Person Class Object

The person class in this assignment will be the parent object class¹³ for the student object class that will be defined later. This class holds the attributes¹⁴ first_name and last_name. Figure 5 shows the class heading and document string:

```
class Person:
    """
    A class to represent a person.

    Properties:
    - first_name (str): The first name of the person.
    - last_name (str): The last name of the person.

    ChangeLog:
    JWidner, 8/11/2024, Created class
    """
```

Figure 5: Person Object Header.

The first function in the person class is used to initialize the attributes using the `__init__` constructor¹⁵ (Figure 6). This function and others in the class objects use the “self” keyword¹⁶ to refer to the object.

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

Figure 6: Person Constructor.

The next function is a type of property known as the “getter” which returns the value held by the attribute¹⁷ (Figure 7). You will notice that the attribute names are preceded with “`__`”. This tells other coders that these variables are private¹⁸, however Python does not enforce this.

```
@property
def first_name(self):
    return self.__first_name.title()
```

Figure 7: first_name Getter.

The next property is known as a “setter” which is used to set the attribute values (Figure 8). This property also includes some error handling that was defined in the previous assignment elsewhere.

```
@first_name.setter
def first_name(self, value: str):
    if value.isalpha() or value == "":
        self.__first_name = value
```

¹³ Root, Randal. Inherited Code, *Mod07-Notes*, p20.

¹⁴ Root, Randal. Data Class Components: Attributes, *Mod07-Notes*, p6.

¹⁵ Root, Randal. Data Class Components: Constructors, *Mod07-Notes*, p6.

¹⁶ Root, Randal. Data Class Components: The Self Keyword, *Mod07-Notes*, p9.

¹⁷ Root, Randal. Adding Data Validation: Properties, *Mod07-Notes*, p15.

¹⁸ Root, Randal. Adding Data Validation: Private Attributes, *Mod07-Notes*, p14.

```

    else:
        raise ValueError("The first name should not contain numbers.")

```

Figure 8: *first_name* Setter.

The next two properties define the “getter” and “setter” for the *last_name* attribute (Figure 9).

```

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.")

```

Figure 9: *last_name* Getter and Setter.

The final function in the *Person* class is used to override the `__str__` method¹⁹ (Figure 10). This is done to allow for a custom return value.

```

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

```

Figure 10: `__str__` override for the *Person* class.

Student Class Object

The next class is also an object, but this one begins with inheriting²⁰ the attributes of the *Person* class by including it as a parameter (Figure 11).

```

class Student(Person):
    """
    A class to represent a student.

    Inherited Properties:
    - first_name (str): The first name of the person.
    - last_name (str): The last name of the person.

    New Properties:
    - course_name (str): The name of the course.

    ChangeLog:
    JWidner, 8/11/2024, Created class
    """

```

Figure 11: *Student* Class Header.

The initialization for this class calls the attributes from the parent class and adds the additional attribute “course name” (Figure 12).

```

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

```

Figure 12: *Student* Class Initialization.

¹⁹ Root, Randal. Inherited Code: Overriding Methods, *Mod07-Notes*, p23.

²⁰ Root, Randal. Inherited Code, *Mod07-Notes*, p20.

The additional attributes is given a “getter” and “setter” just like the other attributes from the previous class (Figure 13).

```
@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:
        raise ValueError("Course name must be a string")
```

Figure 13: *course_name* Getter and Setter.

Finally, we override the `__str__` method to include all three attributes (Figure 14).

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

Figure 14: `__str__` override for the *Student* class.

FileProcessor Class and Associated Functions

The *FileProcessor* class contains two functions that deal with processing JSON files. Each function in this class requires input in the form of parameters²¹ that include the file name and the list variable. Figure 15 displays the class heading and document string:

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

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    JWidner,8/11/2024, Updated Class to store data in Student class
    """
```

Figure 15: *FileProcessor* Class heading and document string

The first function defined as part of this class in the “`read_data_from_file`” function. In this function, we extract data from a JSON file by opening the file in read mode, extract the data using methods imported from the JSON module, and place it inside a list²² (Figure 16). The difference from the last assignment is that the list of dictionaries is converted into a list of “Student” objects. The function then returns the list of Student objects for use elsewhere. This function also includes error handling in the code using a Try-Except block²³. This allows the script to continue to run when errors are encountered. For this code section, we can print an error message to the user if the JSON file is unable to be opened using a class and function defined later in the code.

```
@staticmethod
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

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created function
    JWidner,8/11/2024,added for loop to load json data into a list of student objects
```

²¹ Root, Randal. Parameters, *Mod06-Notes*, p14.

²² Root, Randal. JSON Files: Working with JSON Files, *Mod05-Notes*, p15.

²³ Root, Randal. Structured Error Handling (Try-Except), *Mod05-Notes*, p19.

```

: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 rows.
    for student in list_of_dictionary_data: # Convert the list of dictionary rows into Student objects
        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!", error=e)
except Exception as e:
    IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)

finally:
    if not file.closed:
        file.close()
return student_data

```

Figure 16: Function for reading JSON file.

The second function in this class is used to write data into a JSON file and print out a summary of the data written to the file. This section of code utilizes the concepts of opening, writing data to, and closing a JSON file (Figure 17).²⁴ The data first must be converted back from a list of Student objects to a list of dictionaries. This section also uses error handling in case the data does not correctly save to the JSON file.

```

@staticmethod
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
    JWidner, 8/11/2024, added for loop to add data from student objects to json file

    :param file_name: string data with name of file to write to
    :param student_data: list of dictionary rows to be written 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, indent=1)
        file.close()
        IO.output_student_and_course_names(student_data=student_data)
    except Exception as e:
        message = "Error: There was a problem with writing to the file.\n"
        message += "Please check that the file is not open by another program."
        IO.output_error_messages(message=message, error=e)
    finally:
        if not file.closed:
            file.close()

```

Figure 17: Function for writing to JSON file.

²⁴ Root, Randal. JSON Files: Working with JSON Files, *Mod05-Notes*, p15.

IO Class and Associated Functions

The “IO” Class contains five functions used for receiving input from the user and displaying information to the user. Figure 18 displays the class heading and document string:

```
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
    JWidner,8/11/2024,updated input_student_data to use list of student objects
    """
```

Figure 18: IO Class header

The first function in this class is used for error handling. The error message and error type are input as parameters and the appropriate error message is displayed to the user (Figure 19).

```
@staticmethod
def output_error_messages(message: str, error: Exception = None):
    """ This function displays 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')
```

Figure 19: Error Handling Function.

The next function in this class is used to display the menu to the user. This function receives a string parameter and prints it out (Figure 20).

```
@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.
```

Figure 20: Output Menu function.

The next function asks for the user input and returns the choice they select as a string (Figure 21). If the user enters a choice not recognized by the function, a specific error handling message is raised.

```

@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

```

Figure 21: Menu Choice Function.

The next function outputs messages displaying data showing the first, last, and course name of all students registered for courses. This function receives a list of Student objects and outputs the data in a readable format (Figure 22).

```

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

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

    :param student_data: list of dictionary rows to be displayed

    :return: None
    """
    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)

```

Figure 22: Student Course Output

The final function of this class asks the user to enter data including first name, last name and course name for a student that needs to be registered for a course. The current list with student data is input via a parameter and the updated list is returned (Figure 23). This function includes additional error handling outside of that which is defined in the Person and Student objects.

```

@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

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030, Created function
    JWidner,8/11/2024, updated method to use list of student objects

    :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

```

Figure 23: User Input function.

Main Body of the Script

This script separates the defined functions with the functional part of the program using a concept known as Separation of Concerns (SoC)²⁵. This improves code readability and organization as the code length increases.

Importing Data from the JSON File

The first code in this section uses a pre-defined function from the “FileProcessor” class to import data from the JSON file defined by the “FILE_NAME” constant into a list variable (Figure 24).

```

# 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)

```

Figure 24: Import data from JSON file.

The While Loop

The next section of code establishes a loop using the while loop concept we learned in a previous module²⁶. This tells the script to continue to run the following code while a specific conditional statement is true and presents the menu to the user and asks for a number as a choice (Figure 25). This section of code uses two different functions defined as part of the “IO” class.

```

# Present and Process the data
while True:

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

```

Figure 25: Establish the loop.

Data Entry

If the user selects the first option, the user is given an opportunity to enter data using a function defined in the “IO” class. The data is then saved to a previously defined list (Figure 26).

```

# 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

```

Figure 26: Option 1, data entry.

²⁵ Root, Randal. The Separation of Concerns Pattern, *Mod06-Notes*, p32.

²⁶ Root, Randal. Loops: The while Loop, *Mod03-Notes*, p16.

Presenting Data to the User

If the user chooses option 2, the data that was entered by the user and appended into a list variable is then printed out to the user using a function from the “IO” class. This is a useful troubleshooting technique which helps verify the data once it has been saved to the JSON file (Figure 27).

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

Figure 27: Print out data to the user.

Data File Manipulation

If the user chooses option 3, the data is saved to a JSON file (Figure 28). This is done using a function from the “FileProcessor” class.

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

Figure 28: File Manipulation Code.

Closing the Loop and Ending the Program

The final section of code executes if the user chooses option 4. The loop stops and the program ends (Figure 29). If the user chooses an option not recognized by the conditional statements, error handling defined in the “input_menu_choice” function of the “IO” class.

```
# Stop the loop
elif menu_choice == "4":
    break # out of the loop

print("Program Ended")
```

Figure 29: Final conditional statement to close the loop and end the program.

Outputs

This assignment outputs a menu to the user, asks for input from the user, and outputs the data including any data read in from the JSON file. This assignment also outputs error messages based on exception handling. To test the script, we are asked to run it using both IDE²⁷ and the command console²⁸. For this assignment, PyCharm²⁹ was used as the IDE. The results of these tests can be found below in Figures 30 and 31 respectively.

²⁷ Root, Randal, Integrated Development Environments: IDLE (Integrated Development and Learning Environment), *Mod01-Notes*, p14.

²⁸ Root, Randal, The Command Shell, *Mod01-Notes*, p10.

²⁹ Root, Randal, PyCharm, *Mod03-Notes*, p3.

```
Assignment07 x
D:\Python\Python312\python.exe "E:\Documents\Im

---- 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.
-----

Enter your menu choice number: 2
-----

Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Vic Vu is enrolled in Python 100
-----

---- 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.
-----

Enter your menu choice number: 1
Enter the student's first name: Peter
Enter the student's last name: Parker
Please enter the name of the course: Econ 100

You have registered Peter Parker for Econ 100.

---- 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.
-----

Enter your menu choice number: 3
-----

Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Vic Vu is enrolled in Python 100
Student Peter Parker is enrolled in Econ 100
-----

---- 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.
-----

Enter your menu choice number: 4
Program Ended

Process finished with exit code 0
```

Figure 30: PyCharm Output.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and
PS E:\Documents\Important\UW Python class\_Module07

----- 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.

-----

Enter your menu choice number: 4
Program Ended
PS E:\Documents\Important\UW Python class\_

----- 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.

-----

Enter your menu choice number: 2

Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Vic Vu is enrolled in Python 100

-----

----- 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.

-----

Enter your menu choice number: 1
Enter the student's first name: Peter
Enter the student's last name: Parker
Please enter the name of the course: Econ 100

You have registered Peter Parker for Econ 100.

----- 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.

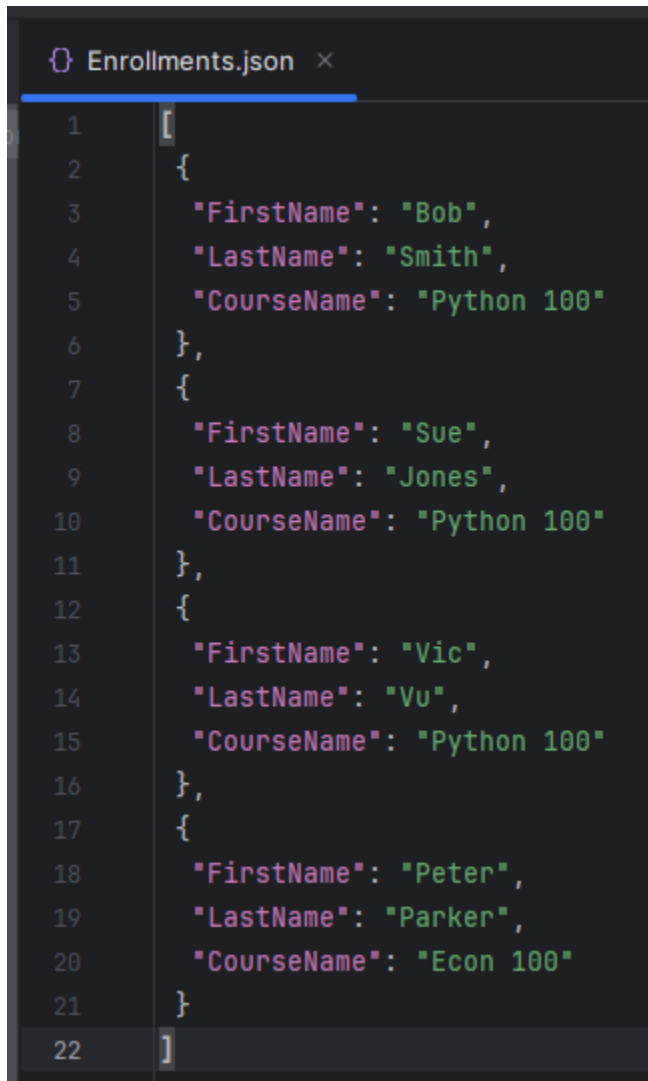
-----

Enter your menu choice number: 3

Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Vic Vu is enrolled in Python 100
Student Peter Parker is enrolled in Econ 100
```

Figure 31: Command Console Output.

I then further verified the functionality of the script by opening the file generated by it using PyCharm (Figure 32).



```
1  [
2    {
3      "FirstName": "Bob",
4      "LastName": "Smith",
5      "CourseName": "Python 100"
6    },
7    {
8      "FirstName": "Sue",
9      "LastName": "Jones",
10     "CourseName": "Python 100"
11   },
12   {
13     "FirstName": "Vic",
14     "LastName": "Vu",
15     "CourseName": "Python 100"
16   },
17   {
18     "FirstName": "Peter",
19     "LastName": "Parker",
20     "CourseName": "Econ 100"
21   }
22 ]
```

Figure 32: Data File Output.

Both the data output to the user and the data found in the JSON file match, confirming that the script works as intended.

Summary

Using the information gained from reviewing the course materials, I was able to successfully complete this assignment and create a python script that reads data from a JSON file and then presents the user with a menu with multiple options and receives input from the user. The functionality of this assignment was the same as assignment 6, the difference is that the data is manipulated using a class object.

The concepts introduced in this assignment were initially difficult to understand, but the demonstration videos and labs help clarify them. I think the use of class objects for data manipulation is interesting and would be useful for more complex programming. The inheritance of class objects sounds more useful for large organizations of similar objects. I liked how this was described in the notes where the Person object was inherited by different kinds of people represented by other objects. I think more information about other useful “magic methods” that are commonly used would be helpful to prepare us for future advance Python classes.