

Renato Felicio
November 24, 2024
IT FDN 110 A
Assignment 07

Creating Python Scripts – Classes and Objects

Introduction

This Assignment 07 involves creating a Python program that utilizes constants, variables, and print statements to display a message about a student's registration for a Python course. Building upon the knowledge gained in Assignment 06, this task introduces new concepts, including the use of data classes.

Preparation for this assignment

To prepare for this assignment, I reviewed the "Module 07 Notes" (Reference 1), completed the three lab examples, and watched both the Module 07 videos available on Washington University's Canvas platform (Figure 1) and the recommended external video in Reference 2. Through these materials, I learnt how to work with data classes, including their attributes, constructors, properties and methods. I also gained an understanding of about the Inheritance concept, where a child class inherit data value, properties and methods from its parent class, and how to override inherited methods.

Mod07 Videos












Module	Name	Link
7	Mod07 - Classes And Functions	https://youtu.be/tLyWwmfu-Vc  
7	Mod07 - Using Constructors	https://youtu.be/7D7damse9xs  
7	Mod07 - Lab01 Review	https://youtu.be/LAEotZSO-AM  
7	Mod07 - Using Properties	https://youtu.be/JE940GjMySI  
7	Mod07 - Lab02 Review	https://youtu.be/L9nN8nMOqr8  
7	Mod07 - Using Inheritance	https://youtu.be/DkBPVcl5POU  
7	Mod07 - Lab03 Review	https://youtu.be/A6ABtRQ-YXs  
7	Mod07 - Pycharm And Github	https://youtu.be/-S7fuwYqHp8  

Figure 1 – Mod07-Videos

Python Scripting

I started by using my script, *Assignment06.py*, from the previous assignment as the foundation for this project. The objective was to implement a set of data classes and objects. First, I created a Person class, which included private attributes for the first and last names, along with "getter" and "setter" properties and error handling. Next, I developed a child class called Student, which inherited the attributes of the Person class and introduced an additional attribute for the course name, complete with its own "getter" and "setter" properties.

After defining these two classes, I updated the methods in the FileProcessor and IO classes to work with objects instead of dictionaries. Finally, I modified the main body of the script to utilize objects, ensuring seamless integration of the new data class structure.

The functions (methods) are organized into four classes: Person, Student, FileProcessor, and IO. The FileProcessor class contains functions for reading data from and writing data to a .json file. The Person and Student classes define the data objects, including the student's first name, last name, and course name. The IO class manages user interactions, such as presenting menu options, gathering input data, displaying messages, and handling error notifications.

The script retains the two global variables defined in the Assignment 06 script: menu_choice, a string used to store the user's selected option, and students, a list used to hold the information of all registered students. These variables are initialized as an empty string and an empty list, respectively.

The script is structured into key sections: the header, imports, variable and constant definitions, class and function definitions, and the main body. Functions are grouped into classes based on their roles, ensuring alignment with the data processing and presentation layers for better organization and clarity.

```
# Header
# Import
# Global Data
# Data Layer
    Definition of data constants
    Definition of variable
# Class Definition
    Data Class
        Class Person
        Class Student
    Processing data layer
        Class FileProcessor
            Function read_data_from_file created
            Function write_data_to_file created
    Presentation data layer
        Class IO
            Function output_error_messages created
            Function output_menu
            Function input_menu_choice
            Function output_student_courses
            Function input_student_data

# Main body of the script
```

The script is displayed in separated figures, split across Figure 2 to Figure 6 according its correspondent sections.

Header, import, constants and variables script parts are presented in the Figure 2.

```

1  # ----- #
2  # Title: Assignment06
3  # Desc: This assignment demonstrates using data classes
4  # with structured error handling and SoC
5  # Change Log: (Who, When, What)
6  # Renato Felicio, 11/16/2024, Created Script
7  # Renato Felicio, 11/23/2024, Modified to work with data classes
8  # <Your Name Here>, <Date>, <Activity>
9  # ----- #
10
11 # Import section
12 import json
13 from typing import TextIO
14
15 # Global Data Layer
16
17 # Define the Data Constants
18 MENU: str = '''
19 ---- Course Registration Program ----
20 Select from the following menu:
21 1. Register a Student for a Course.
22 2. Show current data.
23 3. Save data to a file.
24 4. Exit the program.
25 -----
26 '''
27
28 # Define the Data Constants
29 FILE_NAME: str = "Enrollments.json" # Constant holds the name of the file with students data
30
31 # Define the Data Variables and constants
32 students: list = [] # This variable holds the information of all registered students.
33 menu_choice: str = '' # It holds the user choice.

```

Figure 2 – Python Script Header Import and Variables

Data classes are presented in Figure 3.

```

55 # Class definition
56
57 # Data Class (This section includes person and student data classes)
58
59 @
60 class Person: 1 usage
61     """
62     A class representing person data.
63
64     Properties:
65         student_first_name (str): The student's first name.
66         student_last_name (str): The student's last name.
67
68     ChangeLog:
69         - Renato Felicio, 11/23/2024: Created the class.
70     """
71
72 # Constructor for student's first and last name are defined below:
73 @
74 def __init__(self, student_first_name: str = '', student_last_name: str = ''): # parameters default to empty
75     self.student_first_name = student_first_name # set the attribute using the property to provide validation
76     self.student_last_name = student_last_name # set the attribute using the property to provide validation
77
78 # Getter and Setter Properties for first name are created below
79 @property 4 usages (2 dynamic)
80 def student_first_name(self):
81     return self.__student_first_name.title()
82
83 @student_first_name.setter 3 usages (2 dynamic)
84 def student_first_name(self, value: str):
85     if value.isalpha() or value == '': # checks if user input values are alphabetic characters or empty string
86         self.__student_first_name = value
87     else:
88         raise ValueError("The first name should not contain numbers.") # Custom error message
89
90 # Getter and Setter Properties for last name are created below
91 @property 4 usages (2 dynamic)
92 def student_last_name(self):
93     return self.__student_last_name.title() # checks if user input values are alphabetic characters or empty string
94
95 @student_last_name.setter 3 usages (2 dynamic)
96 def student_last_name(self, value: str):
97     # Method to extract the comma separate data is presented below, it overrides the __str__() method
98     def __str__(self):
99         return f'{self.student_first_name},{self.student_last_name}'
100
101 # Student class is defined below, and it inherited person class
102
103 class Student(Person): 2 usages
104     """
105     A class representing student data.
106
107     Properties:
108         course_name (str): The course name for the student registration.
109
110     ChangeLog: (Who, When, What)
111     Renato Felicio, 11/23/2024, Created Class
112     """
113
114 # Constructor for student's course name is defined below:
115 def __init__(self, student_first_name: str = '', student_last_name: str = '', course_name: str = ''):
116     super().__init__(student_first_name=student_first_name, student_last_name=student_last_name)
117     self.course_name = course_name
118
119 # Getter and Setter Properties for course name are created below
120 @property 4 usages (2 dynamic)
121 def course_name(self):
122     return self.__course_name
123
124 @course_name.setter 4 usages (2 dynamic)
125 def course_name(self, value: str):
126     self.__course_name = value
127
128 # Method to extract the comma separate data is presented below, it overrides the __str__() method
129 @
130 def __str__(self):
131     return f'{self.student_first_name},{self.student_last_name},{self.course_name}'

```

Figure 3 – Python Script Data Layer

Processing data layer is presented in Figure 4.

```

137 # Processing Data Layer
138 class FileProcessor: 3 usages
139     """
140     A collection of processing layer functions that work with json files
141     """
142     ChangeLog: (Who, When, What)
143     Renato Felicio, 11/16/2024, Created Class
144     Renato Felicio, 11/26/2024, Modified Class to work with list of student objects
145     """
146     @staticmethod 1 usage
147     def read_data_from_file(file_name: str, student_data: list): # This function reads data from json file
148         """ This function reads data from a json file into a list of object rows
149         """
150         Notes:
151         - Data sent to the student_data parameter will be overwritten.
152         ChangeLog: (Who, When, What)
153         Renato Felicio, 11/16/2024, Created function
154         Renato Felicio, 11/23/2024, Modified function to work with student data in objects instead of dictionaries
155         """
156         :param file_name: string with the name of the file we are reading
157         :param student_data: List of object rows containing student data
158         :return: List of object rows filled with data
159         """
160         try:
161             file = TextIO = open(file_name, "r") # Open the JSON file for reading
162             json_data: List = json.load(file) # File data is loaded into a table of dictionaries
163             # Now 'json_data' contains the parsed JSON data as a Python List of dictionaries
164             for student in json_data: # This for will convert the student data into a table of objects
165                 student_object = Student(student_first_name=student['first_name'],
166                                         student_last_name=student['last_name'],
167                                         course_name=student['course_name'])
168                 student_data.append(student_object)
169         finally:
170             file.close()
171         except FileNotFoundError as e: # Handles error in case there is no initial file
172             IO.output_error_messages(message="Data file must exist before running this script!", error=e)
173             file = open(FILE_NAME, "a") # Creates an empty initial file, in case of file not found
174             IO.output_error_messages(message="Empty file was created!\n")
175         except Exception as e:
176             IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
177         finally:
178             if file.closed == False:
179                 file.close()
180         return student_data
181
182     @staticmethod 1 usage
183     def write_data_to_file(file_name: str, student_data: list): # This function reads data from json file
184         """ This function writes data to a json file from a list of object rows
185         """
186         ChangeLog: (Who, When, What)
187         Renato Felicio, 11/16/2024, Created function
188         Renato Felicio, 11/23/2024, Modified function to work with students objects instead of dictionaries, and
189         added type exception error
190         """
191         :param file_name: string with the name of the file we are writing to
192         :param student_data: List of object rows containing student data
193         :return: None
194         """
195         try:
196             json_data: list = []
197             for student in student_data: # Converts List of Student objects to list of dictionary rows.
198                 student_json: dict \
199                     = {'first_name': student.student_first_name,
200                       'last_name': student.student_last_name,
201                       'course_name': student.course_name}
202                 json_data.append(student_json)
203             file: TextIO = open(file_name, "w")
204             json.dump(json_data, file) # It writes the list of dictionaries into a json file
205             file.close()
206         except TypeError as e:
207             IO.output_error_messages(message="Please check that the data is a valid JSON format", error=e)
208         except Exception as e: # It handles any exception that could happen when writing the file
209             if file.closed == False:
210                 IO.output_error_messages(message="There was a problem with writing to the file.", error=e)
211                 IO.output_error_messages(message="Please check that the file is not open by another program.", error=e)
212             print()
213         # End of Processing Data Layer

```

Figure 4 – Python Script Processing Data Layer

Presentation data layer and main body are presented in Figure 5 and Figure 6, respectively.

```

137 # Presentation Data Layer
138 class ID: 1 usages
139     """A collection of presentation layer functions that manage user input and output
140     """
141     ChangeLog: (Who, When, What)
142     Renato Felicio, 11/16/2024, Created Class
143     """
144     @staticmethod 9 usages
145     def output_error_messages(message: str, error: Exception = None):
146         """ This function displays a custom error messages to the user
147         """
148         ChangeLog: (Who, When, What)
149         Renato Felicio, 11/16/2024, Created function
150         """
151         :return: None
152         """
153         print(message, end="\n\n")
154         if error is not None:
155             print("-- Technical Error Message --")
156             print(error, error.__doc__, type(error), sep='\n')
157
158     @staticmethod 1 usage
159     def output_menu(menu: str):
160         """ This function displays the menu of choices to the user
161         """
162         ChangeLog: (Who, When, What)
163         Renato Felicio, 11/16/2024, Created function
164         """
165         :return: None
166         """
167         print(menu)
168
169     @staticmethod 1 usage
170     def input_menu_choice():
171         """ This function gets a menu choice from the user
172         """
173         ChangeLog: (Who, When, What)
174         Renato Felicio, 11/16/2024, Created function
175         """
176         :return: string with the users choice
177         """
178         choice = "0"
179         try:
180             choice: str = input("What would you like to do: ")
181             if choice not in ("1", "2", "3", "4"): # Note these are strings
182                 raise Exception("Please, choose only 1, 2, 3, or 4")
183         except Exception as e:
184             IO.output_error_messages(e.__str__()) # Not passing e to avoid the technical message
185         return choice
186
187     @staticmethod 1 usage
188     def output_student_courses(student_data: list):
189         """ This function displays the current data to the user
190         """
191         ChangeLog: (Who, When, What)
192         Renato Felicio, 11/16/2024, Created function
193         Renato Felicio, 11/23/2024, Modified to work with objects
194         """
195         :return: None
196         """
197         # Process the data to create and display a custom message
198         print("--" * 50)
199         #student_data=
200         for student in student_data:
201             print(f"Student {student.student_first_name} "
202                   f"{student.student_last_name} is enrolled in {student.course_name}")
203             print("--" * 50)
204
205     @staticmethod 1 usage
206     def input_student_data(student_data: list):
207         """ This function gets data from the user and adds it to a list of object rows
208         """
209         ChangeLog: (Who, When, What)
210         Renato Felicio, 11/16/2024, Created function
211         Renato Felicio, 11/23/2024, Modified function to work with objects instead of dictionaries
212         """
213         :param student_data: List of dictionary rows containing student current data
214         :return: List of object rows filled with a new row of data
215         """
216         try:
217             # Input of data
218             student = Student()
219             student.student_first_name: str = input("Enter the student's first name: ") # Holds student first name input
220             student.student_last_name: str = input("Enter the student's last name: ") # Holds student last name input
221             student.course_name: str = input("Please enter the name of the course: ") # Holds course name input
222             student_data.append(student)
223         except ValueError as e:
224             IO.output_error_messages(message="", error=e)
225         except Exception as e:
226             IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
227         return student_data
228
229     # End of Presentation Data Layer
230
231     # End of class Definition

```

Figure 5 – Python Script Presentation Data Layer

```

3291
3292     # Start of the main body of the script
3293
3294     # Read data from a file
3295     students:list = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
3296
3297     while (True): # Loops through the menu of options
3298         # Present the menu of choices
3299         IO.output_menu(MENU)
3300         menu_choice=IO.input_menu_choice()
3301
3302         # Input user data
3303         if menu_choice == "1": # This will not work if it is an integer!
3304             students=IO.input_student_data(students)
3305             continue
3306
3307         # Present the current data
3308         elif menu_choice == "2":
3309             # Process the data to create and display a custom message
3310             IO.output_student_courses(student_data=students)
3311             continue
3312
3313         # Save the data to a file and present to user
3314         elif menu_choice == "3":
3315             FileProcessor.write_data_to_file( file_name: FILE_NAME, student_data: students)
3316             IO.output_student_courses(students)
3317             continue
3318
3319         # Stop the loop
3320         elif menu_choice == "4":
3321             break # out of the loop
3322         else:
3323             print("Please only choose option 1, 2, or 3")
3324
3325     print("Program Ended")
3326
3327

```

Figure 6 – Python Script Main Body

Python Script Testing

I executed the Python script using PyCharm (see Figure 8) and also tested it in the Windows Command Prompt (see Figure 9), verifying that the script performed as expected in both environments. Additionally, I confirmed that the Enrollments.json file was updated correctly and contained the expected output (see Figure 10). Following the assignment instructions, I used a initial Enrollments.json. The original content of this file is shown in Figure 7.

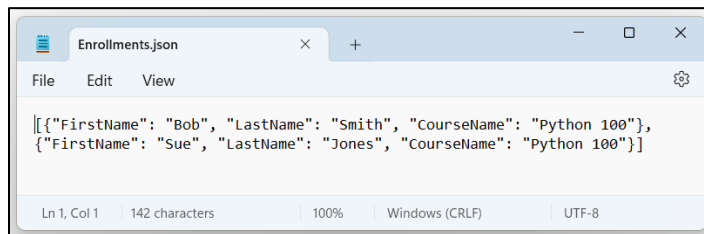


Figure 7 – Initial Enrollment File Python



Figure 8 – Python Script PyCharm Run

```
Command Prompt
D:\UW\IT FDN 110 A\Module07-1\Module07\Assignment\Assignment07>python Assignment07.py

---- 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.
-----

What would you like to do: 1
Enter the student's first name: Michael
Enter the student's last name: Scott
Please enter the name of the course: 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.
-----

What would you like to do: 1
Enter the student's first name: Dwight
Enter the student's last name: Schrute
Please enter the name of the course: 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.
-----

What would you like to do: 2
-----
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Michael Scott is enrolled in Python 100
Student Dwight Schrute 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.
-----

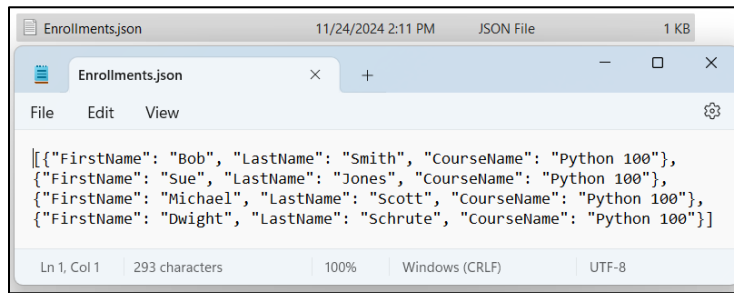
What would you like to do: 3
-----
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Michael Scott is enrolled in Python 100
Student Dwight Schrute is enrolled in Python 100
-----

---- 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.
-----

What would you like to do: 4
Program Ended
```

Figure 9 – Python Script Windows Command Prompt Run

Figure 10 shows that the Enrollments.json file was updated correctly with the two new student data.



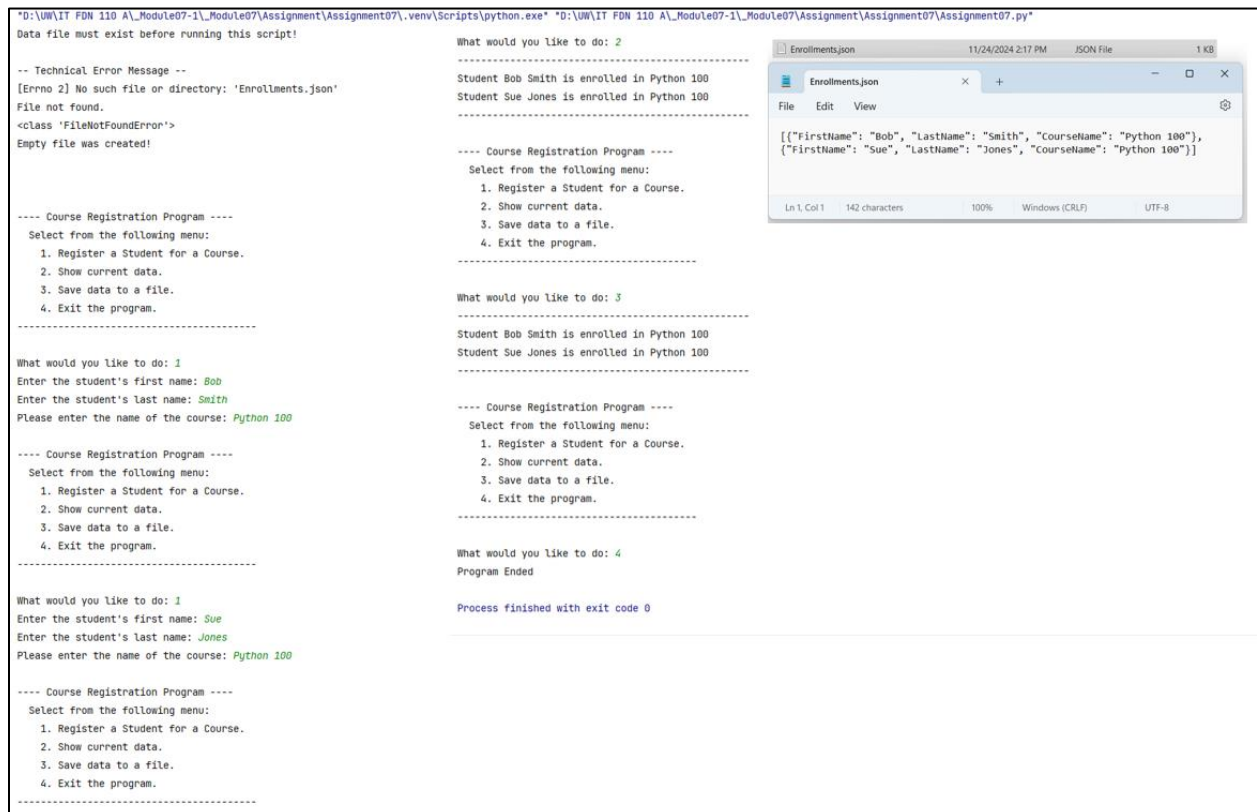
```
Enrollments.json 11/24/2024 2:11 PM JSON File 1 KB
Enrollments.json
File Edit View
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"},
{"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"},
{"FirstName": "Michael", "LastName": "Scott", "CourseName": "Python 100"},
{"FirstName": "Dwight", "LastName": "Schrute", "CourseName": "Python 100"}]
Ln 1, Col 1 293 characters 100% Windows (CRLF) UTF-8
```

Figure 10 – Output JSON File Content

Python Script Error Handling Test

After testing the script with valid user inputs using a pre-existing Enrollments.json file, I evaluated its error-handling capabilities for scenarios such as a "file not found" exception and invalid first and last name inputs. All tests were conducted using PyCharm.

The Figure 11 below shows the FileNotFoundError handling and the Enrolments.json being updated correctly after menu choice 3 is selected.



```
"D:\UW\IT_FDN_110_A_Module07-1\Module07\Assignment\Assignment07\.venv\Scripts\python.exe" "D:\UW\IT_FDN_110_A_Module07-1\Module07\Assignment\Assignment07\Assignment07.py"
Data file must exist before running this script!

-- Technical Error Message --
[Errno 2] No such file or directory: 'Enrollments.json'
File not found.
<class 'FileNotFoundError'>
Empty file was created!

---- 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.

What would you like to do: 1
Enter the student's first name: Bob
Enter the student's last name: Smith
Please enter the name of the course: 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.

What would you like to do: 1
Enter the student's first name: Sue
Enter the student's last name: Jones
Please enter the name of the course: 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.

What would you like to do: 2
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100

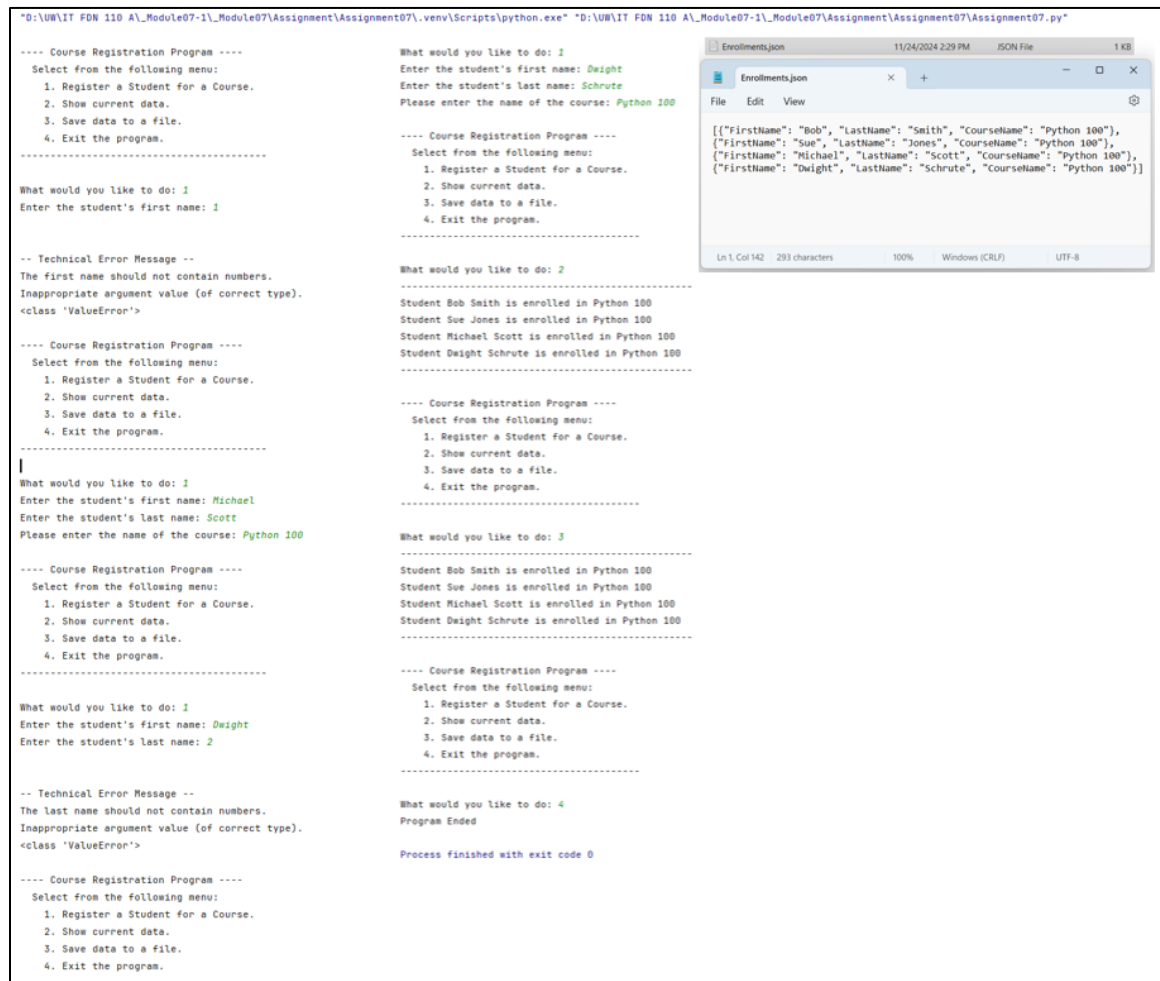
---- Course Registration Program ----
Select from the following menu:
1. Register a Student for a Course.
2. Show current data.
3. Save data to a file.
4. Exit the program.

What would you like to do: 3
Program Ended

Process finished with exit code 0
```

Figure 11 – FileNotFoundError Handling

The second error handling test involved checking the condition where the user enters non-alphabetic characters for the first or last names (see Figure 12). The Enrollments.json file was also correctly updated



The screenshot displays a terminal window on the left and a file editor on the right. The terminal shows the execution of a Python script named 'Assignment07.py'. The script is a 'Course Registration Program' that prompts the user to select from a menu (1. Register a Student for a Course, 2. Show current data, 3. Save data to a file, 4. Exit the program). It then prompts for student details (first name, last name, and course name). The terminal shows the user entering 'Dwight' for the first name, 'Schrute' for the last name, and 'Python 100' for the course name. The program then displays a list of enrolled students: Bob Smith, Sue Jones, Michael Scott, and Dwight Schrute, all enrolled in Python 100. The program ends with the message 'Program Ended' and 'Process finished with exit code 0'. The file editor on the right shows the 'Enrollments.json' file, which contains a JSON array of student records: [{"firstName": "Bob", "lastName": "Smith", "courseName": "Python 100"}, {"firstName": "Sue", "lastName": "Jones", "courseName": "Python 100"}, {"firstName": "Michael", "lastName": "Scott", "courseName": "Python 100"}, {"firstName": "Dwight", "lastName": "Schrute", "courseName": "Python 100"}].

```
"D:\UW\IT FDN 110 A\Module07-1\Module07\Assignment\Assignment07\Assignment07.py"
---- 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.
-----
What would you like to do: 1
Enter the student's first name: Dwight
Enter the student's last name: Schrute
Please enter the name of the course: 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.
-----
What would you like to do: 2
-----
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Michael Scott is enrolled in Python 100
Student Dwight Schrute 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.
-----
What would you like to do: 3
-----
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Michael Scott is enrolled in Python 100
Student Dwight Schrute 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.
-----
What would you like to do: 4
Program Ended
Process finished with exit code 0

-- Technical Error Message --
The first name should not contain numbers.
Inappropriate argument value (of correct type).
<class 'ValueError'>

---- 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.
-----
What would you like to do: 1
Enter the student's first name: Michael
Enter the student's last name: Scott
Please enter the name of the course: 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.
-----
What would you like to do: 1
Enter the student's first name: Dwight
Enter the student's last name: 2

-- Technical Error Message --
The last name should not contain numbers.
Inappropriate argument value (of correct type).
<class 'ValueError'>

---- 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.
-----
```

```
{
  "firstName": "Bob", "lastName": "Smith", "courseName": "Python 100",
  "firstName": "Sue", "lastName": "Jones", "courseName": "Python 100",
  "firstName": "Michael", "lastName": "Scott", "courseName": "Python 100",
  "firstName": "Dwight", "lastName": "Schrute", "courseName": "Python 100"
}
```

Figure 12 – Non-Alphabetic Characters Error Handling

GitHub

Script and documentation for this assignment is available in my GitHub site:

<https://github.com/rfnaval/IntroToProg-Python-Mod07.git>

Summary

This assignment built upon the previous one, providing an opportunity to learn and practice key Python concepts such as data classes, their private attributes, constructors, “self” keyword, “getter” and “setter” properties, the inheritance concept, and overriding methods. It was very interesting to work with tables (list of lists) containing objects instead of dictionaries, and to modify the FileProcessor and IO classes, along with the script main body to work with objects as well.

References

1. Module 07 - Classes and Objects, Randal Root, January 02, 2024.
2. External site: [Python OOP Tutorial 1: Classes and Instances](#), Corey Schafer.