Name: Sohail Nassiri

Date: September 14th, 2024

Course: IT FDN 110 B Su 24 - Foundations of Programming: Python

GitHub URL: https://github.com/jnts221/IntroToProg-Python-Mod07

Assignment 07 – Creating a Python Script Using Object Oriented Programming and Structured Error Handling

Introduction

This assignment requires the use of constants, variables, and a variety of functions, in a Python script. In addition, it incorporates the use of string formatting, while and for loops, programming menus, conditional logic, classes, object-oriented programming, exception handling and java script-oriented notation (JSON) files to work with data. The purpose of the Python script is to allow users to enter and display multiple registrations for students and their associated courses along with the ability to read and save the data to/from a JSON file if desired.

Creating the Python Script

Using the PyCharm IDE, a Python script is created. The code begins with the script header, as shown in **Figure 1**, in order to give some background information on the purpose of the script. In this case, this script is being written in order to complete the requirements of Assignment 7 demonstrating the use of important concepts in Module 07 being:

- Classes
- Objects
- Functions
- Constructors
- Properties
- Inheritance

```
# ------------------------------------------------------------------------------------ #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
#   Sohail Nassiri,09/09/2024,Created Script
# ------------------------------------------------------------------------------------ #
```

Figure 1. Script Header

First, the Main section is started by importing the JSON module since the script will be utilizing JSON files. Next, the data constants are defined (see **Figure 2**), which are designated in upper case. Also, type hints are used to allow the reader to know the data type associated with each constant or variable. The following constants are assigned:

- The course registration program menu (see below), is assigned to MENU (string).

---- 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

- The name of the .json file, 'enrollments.json', is assigned to FILE_NAME (string)

```
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"  # Set the json file name
```

Figure 2. Defining the Data Constants

The global data variables, which are declared outside of the functions and accessible anywhere within the script, are defined and set to empty strings as follows (see **Figure 3**):

- students (list) is set to an empty list
- menu_choice (string) is set to empty string

```
# Define the Data Variables
students: list = []  # Table of student data
menu_choice: str = ''  # Hold the choice made by the user
```

Figure 3. Defining the Data Variables

Then, the following four classes are created in order to organize the code using object-oriented programming (see **Figures 4a-4e**):

- Person – a class representing person data
  - The class contains the properties first_name (str) and last_name (str) to represent each person's first and last name.
  - A constructor, which is a way to create objects with an initial set of values, is created using __init__ with private attributes for first_name and last_name. This is also known as an instance method.

    - Properties are used to get and set the instance variables. In this case, its use is for additional type checking and logic.
    - The setter contains the validation check to ensure the person's first and last name entered is alphabetical. During this check, the first_name and last_name variables are secret (denoted by the "self.__") and cannot be accessed outside of the class.
    - Ultimately, the constructor inside the Person class jumps to the setters and runs the desired validation steps prior to assigning the user's input to the first_name and last_name variables.
- Student – a class representing student data
    - The class contains the properties first_name (str), last_name (str) and course_name (str). Also, it inherits code from the Person class, known as inheritance. Inheritance allows for increased flexibility and more concise code since it prevents the need to duplicate information between the Person and Student classes.
    - The super().__init__ constructor is used to pass the parameter data to the Person "super" or "parent" class. Thus, the logic setters used in the Person class do not need to be duplicated in the Student class since they are already occurring.
    - The same approach of constructors, properties, and setters are used for the course_name. The constructor inside the Student class jumps to the setters prior to assigning the user's input to course_name.
    - The __str__() method is overridden to return the full student data being the first name, last name and course name.
- FileProcessor – a class that is a collection of processing layer functions that work with json files
    - The read_data_from_file function is defined with parameters (local variables) file_name (str) and student_data (list). This function reads data from a .json file into a list of dictionary rows.
        - The file is opened using the open() function and read using "r" mode.
        - The json.load(file) function parses the data into a list of dictionary rows.
        - A for loop is used to iterate through each row in the list of dictionary data and run it through the Student class before assigning them as first-class objects to student_object (list of student object rows).
        - The student_data variable is appended with data from student_object.
        - A try-except-finally block is added within the code to serve the following functions:
        - Try – Attempts to run the code in the respective block.
        - Except – Runs if exception occurs in the Try block to allow the user to know the cause of the error. In this case, the specific exception is the FileNotFoundError if the .json file is not found. A general Exception error is added as well in case any other exception that is not specifically called out arises.
        - Finally – Runs regardless of whether code successfully executes or if exception is raised to ensure a specific action is carried out. In this case, it is to make sure the .json file is closed before performing any further actions.
    - The write_data_to_file function is defined with parameters file_name (str) and student_data (list). This function writes data to the 'enrollments.json' file from a list of dictionary rows.
        - First, a for loop is used to iterate over the student table and cast it to a list of dictionaries. Next, the file is opened using the open() function. In this case, the file is opened using write mode ('w'). Then, the .json file is written with the content of the list_of_dictionary_data variable using the json.dump function. Finally, the file is saved and closed using the close() function. Similar to when

the .json file is read earlier in the script, a try-except-finally block is used in case the file is not found.
        - Thus, the current data collected by the user is written to the .json file.
- IO – a class that is a collection of presentation layer functions that manage user input and output
    - The output_error_messages function is defined with parameters message (str) and error (Exception) with the argument of None. This function displays custom error messages to the user.
    - The output_menu function is defined with parameter menu (str). This function displays the course registration program menu to the user.
    - The input_menu_choice function is defined. This function gets a menu choice from the user.
    - The output_student_courses function is defined with parameter student_data (list). This function displays the current data to the user using a formatted string.
    - The input_student_data function is defined with parameter student_data (list). This function gets data from the user and adds it to a list of dictionary rows by first passing the Student class to the student variable and receiving the user inputs for the student's first name, last name and course name. The student_data variable is then appended with the data stored in the student variable.

```python
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:
        - Sohail Nassiri, 09.09.2024: Created the class.
    """

    # Constructor with private attributes for the first_name and last_name data
    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name

    #  Property getter and setter for first name using the same code as in the Student class
    @property  # Decorator for the getter or accessor
    def first_name(self):
        return self.__first_name.title()  # Formatting code

    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha() or value == "":  # Is character or empty string
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")

    # Property getter and setter for last name using the same code as in the Student class
    @property
    def last_name(self):
        return self.__last_name.title()  # Formatting code

    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha() or value == "":  # Is character or empty string
            self.__last_name = value
        else:
            raise ValueError("The last name should not contain numbers.")

    #  Override the default __str__() method's behavior and return a coma-separated string of data
    def __str__(self):
        return f'{self.first_name},{self.last_name}'
```

Figure 4a. Creating Classes Using Object Oriented Programming (Constructors, Properties and Inheritance) - Person

```python
#  Student class which will inherit code from the person class
class Student(Person):
    """
    A class representing student data.

    Properties:
        first_name (str): The student's first name.
        last_name (str): The student's last name.
        course_name (str): The course name that the student is registered for.

    ChangeLog: (Who, When, What)
    Sohail Nassiri,09.09.2024,Created Class, added properties, private attributes, moved first_name and last_name into a
    parent class
    """

    def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
        #  Passing the parameter data to the Person "super" class
        super().__init__(first_name=first_name, last_name=last_name)
        self.course_name = course_name

    #  Assignment to the course_name property using the course_name parameter
    4 usages (2 dynamic)
    @property
    def course_name(self):
        return self.__course_name

    #  Getter and setter for course_name

    4 usages (2 dynamic)
    @course_name.setter
    def course_name(self, value: str):
        self.__course_name = value

    #  Overriding the __str__() method to return the student data
    def __str__(self):
        return f'{self.first_name},{self.last_name},{self.course_name}'
```

Figure 4b. Creating Classes Using Object Oriented Programming (Constructors, Properties and Inheritance) - Student

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

    ChangeLog: (Who, When, What)
    Sohail Nassiri,09.09.2024,Created Class
    """

    1 usage
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ This function reads data from a json file into a list of dictionary rows

        Note:
        - Data sent to the student_data parameter will be overwritten.

        ChangeLog: (Who, When, What)
        Sohail Nassiri,09.09.2024,Created function

        :param file_name: string with the name of the file we are reading
        :param student_data: list of dictionary rows we are adding data to
        :return: list of dictionary rows filled with data
        """

        try:
            file = open(file_name, "r")
            list_of_dictionary_data = json.load(file)
            for student in list_of_dictionary_data:
                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:  # Raises exception if file is not found
            # Sending error messages to a function in IO
            IO.output_error_messages( message: "Text file must exist before running this script!", e)
        except Exception as e:  # Raises any other general exception that is not specifically called out
            IO.output_error_messages( message: "There was a non-specific error when reading the file!", e)
        finally:
            if file:
                file.close()
        return student_data
```

Figure 4c. Creating Classes Using Object Oriented Programming (Constructors, Properties and Inheritance) – File Processor

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

        ChangeLog: (Who, When, What)
        Sohail Nassiri,09.09.2024,Created function

        :param file_name: string with the name of the file we are writing to
        :param student_data: list of dictionary rows we have in our data
        :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()
            IO.output_student_courses(student_data=student_data)  # Sending output to a function in IO
        except FileNotFoundError as e:  # Raises exception if file is not found
            # Sending error messages to a function in IO
            IO.output_error_messages( message: "Text file must exist before running this script!", e)
        except Exception as e:  # Raises any other general exception that is not specifically called out
            IO.output_error_messages( message: "There was a non-specific error when reading the file!", e)
        finally:
            if file:
                file.close()
```

Figure 4d. Creating Classes Using Object Oriented Programming (Constructors, Properties and Inheritance) – File Processor

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

    ChangeLog: (Who, When, What)
    Sohail Nassiri,09.09.2024,Created Class, Added menu input/output functions, displaying of data, and custom error
    messages
    """

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

          Note: Allows to customize error messages in one place and affect all error handling

        ChangeLog: (Who, When, What)
        Sohail Nassiri,09.09.2024,Created function and toggling technical message off if no exception object is passed

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

    1 usage
    @staticmethod
    def output_menu(menu: str):
        """ This function displays a menu of option to the user

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

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

        :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 4e. Creating Classes Using Object Oriented Programming (Constructors, Properties and Inheritance) – IO

```python
2 usages
@staticmethod
def output_student_courses(student_data: list):
    """ This function displays the current data to the user

    :return: None
    """
    print("-" * 50)
    for student in student_data:  # Iterates through each row of table
        print(
            f"Student {student.first_name} {student.last_name} is enrolled in {student.course_name}")
    print("-" * 50)

1 usage
@staticmethod
def input_student_data(student_data: list):
    """ This function gets data from the user and adds it to a list of dictionary rows

    :param student_data: list of dictionary rows containing our current data
    :return: list of dictionary rows filled with a new row of data
    """

    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(
            f"You have registered {student.first_name} {student.last_name} for {student.course_name}.")  # Displays
        #  input registration
    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 4f. Creating Classes Using Object Oriented Programming (Constructors, Properties and Inheritance) - IO

Lastly, the main body of the script is written where the classes and functions are called to carry out the desired outcome. The user is prompted to select from the course registration program menu. The options allow the user to enter multiple registrations for students and their courses, display the data collected, read and save data to/from a json file, and/or exit the program. In addition, exception handling is incorporated in order to guide the user through possible errors that may be encountered. This is carried out as follows (see **Figure 4g**):

- When the program begins, the data in “enrollments.json” is automatically read into a two-dimensional table (a list of dictionary rows) by calling the FileProcessor class, read_data_from_file function and passing the FILE_NAME and students arguments to the respective parameters. The read_data_from_file function then converts the table to a list of student object rows and stores it in the student_object variable and appends that to the student_data variable.

- A while loop is set to true, which results in an infinite loop until exited by a break. Inside the while loop, the course registration program menu is printed and the user is asked what selection they would like to make by calling the IO class, output_menu and input_menu_choice functions.
- Conditional logic is used, specifically if, else-if (elif) and else statements, to carry out the proper action based on the selection made:
    - If the user selects “1”, they are prompted to enter the student’s first and last name along with the course name. The data collected is displayed and added to the two-dimensional table. In addition, if a non-alphabetical name is entered for the first and last name, a ValueError is raised to notify the user and prompt them to try again. This is achieved by calling the IO class and input_student_data function, which references the Student class, and passing the students argument to the student_data parameter.
    - If the user selects “2”, the current data collected is displayed. This is done by calling the IO class and output_student_courses function and passing the students argument to the student_data parameter.
    - If the user selects “3”, the current data collected by the user is written to the ‘enrollments.json’ file. This is done by calling the FileProcessor class and write_data_to_file function and passing the FILE_NAME and students arguments to the respective parameters.
    - If the user selects “4”, the while loop is exited by introducing a break. In addition, this ends the program and the user is notified with a print statement.
    - If the user enters in an invalid selection, the user is notified with a print statement and prompted to select from the menu again. This is handled by the input_menu_choice function in the IO class.
- As a result of the infinite while loop, the user will continuously be prompted to make selections from the menu until the choice of exiting the program is made.

```python
# 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_courses(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

print("Program Ended")
```

Figure 4g. Main Body of Script

Running the Python Script

To start, when running the script, the user is prompted to select from the course registration program menu. **Figures 5a – 5b** and **Figures 6a – 6b** show what occurs with each selection as described in the "Creating the Python Script" section when running in the PyCharm IDE and command prompt, respectively. This example includes if the user enters a non-alphabetical name and multiple student registrations. When properly running the script, to prevent an error from occurring when reading the enrollments .json file, the following starting data was entered: Vic,Vu,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: J4
One of the values was the correct type of data!

-- 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.
----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: John
Enter the student's last name: D4
One of the values was the correct type of data!

-- 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.
----------------------------------------

Enter your menu choice number: 1
Enter the student's first name: John
Enter the student's last name: Doe
Please enter the name of the course: Math 100
You have registered John Doe for Math 100.
```

Figure 5a. Running the Python Script from the PyCharm IDE

```
---- 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 Vic Vu is enrolled in Python 100
Student John Doe is enrolled in Math 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 Vic Vu is enrolled in Python 100
Student John Doe is enrolled in Math 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 5b. Running the Python Script from PyCharm IDE

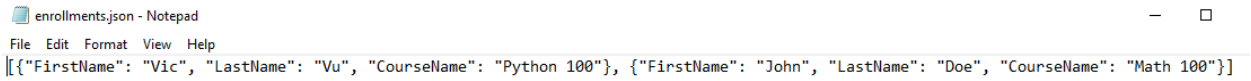
```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: J4
One of the values was the correct type of data!

-- 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.
----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: John
Enter the student's last name: D4
One of the values was the correct type of data!

-- 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.
----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: John
Enter the student's last name: Doe
Please enter the name of the course: Math 100
You have registered John Doe for Math 100.
```

Figure 6a. Running the Python Script from the Command Prompt

```
---- 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 Vic Vu is enrolled in Python 100
Student John Doe is enrolled in Math 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 Vic Vu is enrolled in Python 100
Student John Doe is enrolled in Math 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
```

Figure 6b. Running the Python Script from the Command Prompt

As shown below in **Figure 7**, when selecting Option #3, the data is written and saved to the 'enrollments.json' file.



[{"FirstName": "Vic", "LastName": "Vu", "CourseName": "Python 100"}, {"FirstName": "John", "LastName": "Doe", "CourseName": "Math 100"}]

Figure 7. Data Stored in .JSON File After Running the Python Script

Summary

The goal of this assignment was to demonstrate the creation of a Python script that prompts the user to select from a course registration program menu in order to allow for entering and displaying multiple registrations for students and their associated courses along with the ability to read data from/save data to a JSON file using object-oriented programming and the separation of concerns method. Constants, variables, classes, various functions, string formatting, while and for loops, programming menus, conditional logic, classes, objects, constructors, properties, inheritance, exception handling and JSON files are utilized in this script in order to achieve the desired outcome.