Andrew Yarberry

11/29/2024

ITD FDN 110B Foundations of Programming: Python

Module 07

https://github.com/Finviel314/Python110

Assignment 07 Further Explorations in Object-Oriented Programming

Introduction

For assignment seven, continue to refactor our program by introducing person and student objects. The core function will remain unchanged, but we will continue to parameterize our code into modular blocks.

Person and Student Class

The first step was to create a new class called ‘Person’ which will handle the first and last name input by the user as student objects. “Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for creating objects.” (1) By creating a class for a person we take inputs from the user and pass the information to be stored as a person object inside our program.

```
class Person:  1 usage
    def __init__(self, first_name: str = '', last_name: str = ''):
        """
        Initializes a Person object.

        Args:
            first_name (str, optional): The person's first name. Defaults to an empty string.
            last_name (str, optional): The person's last name. Defaults to an empty string.
        """
        self.first_name = first_name
        self.last_name = last_name
```

Figure 1: Person Class Example

The __init__ is a default constructor method in Python that is used to initialize objects for a class, in this case the Person class. We also utilize the keyword ‘self’ which is used to access instance data or functions.

Additionally, we need to employ getter and setter methods to collect and mutate data while ensuring that data is encapsulated.

- **Getter:** A method that allows you to *access* an attribute in a given class
- **Setter:** A method that allows you to *set* or *mutate* the value of an attribute in a class

Figure 2: Person Class Example (2)

In my code I use the '@property' decorator which can be used as getter. Additionally I used '.title' which is a built in method that capitalizes the first letter in a string. It is useful for formatting names or other proper nouns.

```
@property  3 usages (2 dynamic)
def first_name(self):
    """
    Gets the person's first name.

    Returns:
        str: The person's first name, capitalized.
    """
    return self._first_name.title()  # Capitalizes first letter of name
```

Figure 3: Getter Example for Person Class

Additionally, I incorporated our data validation in the setter method to verify that the person object for first name does not contain numbers or special characters. There is a similar block of code for the last name.

```
@first_name.setter  3 usages (2 dynamic)
def first_name(self, value: str):
    """
    Sets the person's first name.

    Args:
        value (str): The new first name.

    Raises:
        ValueError: If the new first name contains non-alphabetic characters.
    """
    if value.isalpha() or value == '':  # checks if alpha or number
        self._first_name = value
    else:
        raise ValueError("The last name should not contain numbers.")
```

Figure 4: Setter Example for Person Class

Student

Next, we need a class to handle the information for a student. This will take the data from a previously created person class and mutate it to include a course that the student is registering for.

```
class Student(Person):  3 usages
    def __init__(self, first_name: str = '', last_name: str = '', course: str = ''):
        """
        Initializes a Student object.

        Args:
            first_name (str, optional): The student's first name. Defaults to an empty string.
            last_name (str, optional): The student's last name. Defaults to an empty string.
            course (str, optional): The student's course. Defaults to an empty string.
        """
        self._course = course
        super().__init__(first_name=first_name, last_name=last_name)
```

Figure 5: Student Class Example

This class is child to the person class and inherits the information contained in the parent class. We can use the 'super ()' function which can refer to the parent class. In this case we get the objects stored for first and last name. My script uses similar code for getters and setters for input course information. We learned about the __str__ method to return a string from this class as seen in Figure 6.

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

Figure 6: Return String Example

It is interesting to note that the code will function fine without the line of code shown in Figure 6. It will output fine to JSON. The only distinguishing difference between using and not using the code in Figure 6 is how the data is handled. The variable student data appears to be a list of objects (Figure 7) where with the format code the variable becomes a list of dictionaries (Figure 8).

```
file = {TextIOWrapper} <_io.TextIOWrapper name='Enrollments.json' mode='r' encoding='cp1252'>
file_name = {str} 'Enrollments.json'
list_of_dict = {list: 4} [{'CourseName': 'phython 100', 'FirstName': 'Andrew', 'LastName': 'Yarberry'}, {'CourseName': 'ad', 'FirstName': 'Ad', 'LastName': 'Ad'}, {'CourseName': 'P... View
student = {dict: 3} {'CourseName': 'Python 100', 'FirstName': 'Vic', 'LastName': 'Vu'}
student_data = {list: 4} [<__main__.Student object at 0x0000021DE445E660>, <__main__.Student object at 0x0000021DE43B74D0>, <__main__.Student object at 0x0000021... View
  0 = {Student} <__main__.Student object at 0x0000021DE445E660>
  1 = {Student} <__main__.Student object at 0x0000021DE43B74D0>
  2 = {Student} <__main__.Student object at 0x0000021DE43B7890>
  3 = {Student} <__main__.Student object at 0x0000021DE4402190>
  __len__ = {int} 4
  Protected Attributes
student_object = {Student} <__main__.Student object at 0x0000021DE4402190>
```

Figure 7: Debugger Without __Str__ Format Code

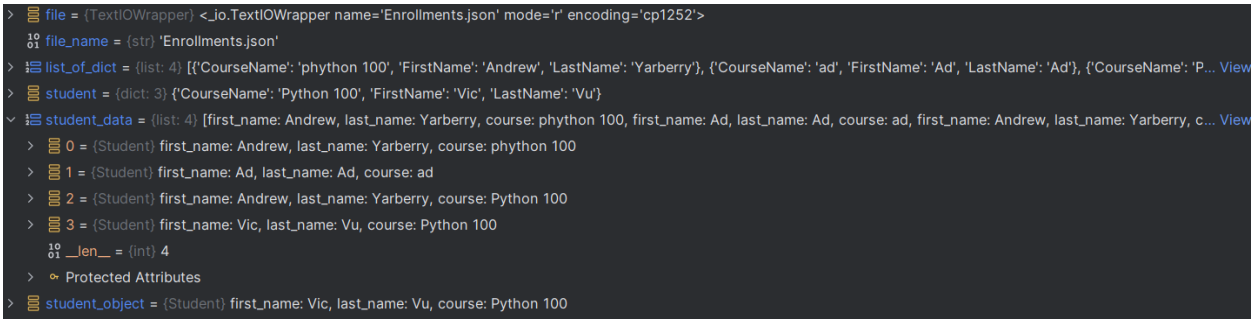

Figure 8: Debugger With __Str__ Format Code

This appears to format student object into a dictionary and brings this back into alignment with where the code was prior to refactor but is not more encapsulated due to the addition of the 'Person' and 'Student' classes.

Testing the Program

I was able to successfully open and run the program in both PyCharm and CMD.

Example of selection one from menu to input new student. Code has data validation to ensure the name is alphabetic. Code prints current data to the console.

```
Enter your menu choice number: 1
Enter the students first name: 2
The last name should not contain numbers.
Enter the students first name: Andrew
Enter the students last name: Yarberry
Enter the students course: Python 100

Here is the current data:
-------------------------------------------------
Andrew Yarberry phython 100
Ad Ad ad
Andrew Yarberry Python 100
-------------------------------------------------
```

Figure 9: Student Input Example

Selection two prints the current data to the console and looks very similar to the output in selection one.

```
Enter your menu choice number: 2
-------------------------------------------------
Andrew Yarberry phython 100
Ad Ad ad
Andrew Yarberry 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
----------------------------------------
```

Figure 10: Current Data Output Example

Program will successfully save the current data to file in JSON format.

```
Enter your menu choice number: 3
The following data was saved to file:
-------------------------------------------------
Andrew Yarberry phython 100
Ad Ad ad
Andrew Yarberry 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
----------------------------------------
```

Figure 11: Save Data to File Example

Assignment07.py Enrollments.json


```
[
    {
        "FirstName": "Andrew",
        "LastName": "Yarberry",
        "CourseName": "phython 100"
    },
    {
        "FirstName": "Ad",
        "LastName": "Ad",
        "CourseName": "ad"
    },
    {
        "FirstName": "Andrew",
        "LastName": "Yarberry",
        "CourseName": "Python 100"
    }
]
```


Figure 11: JSON Output

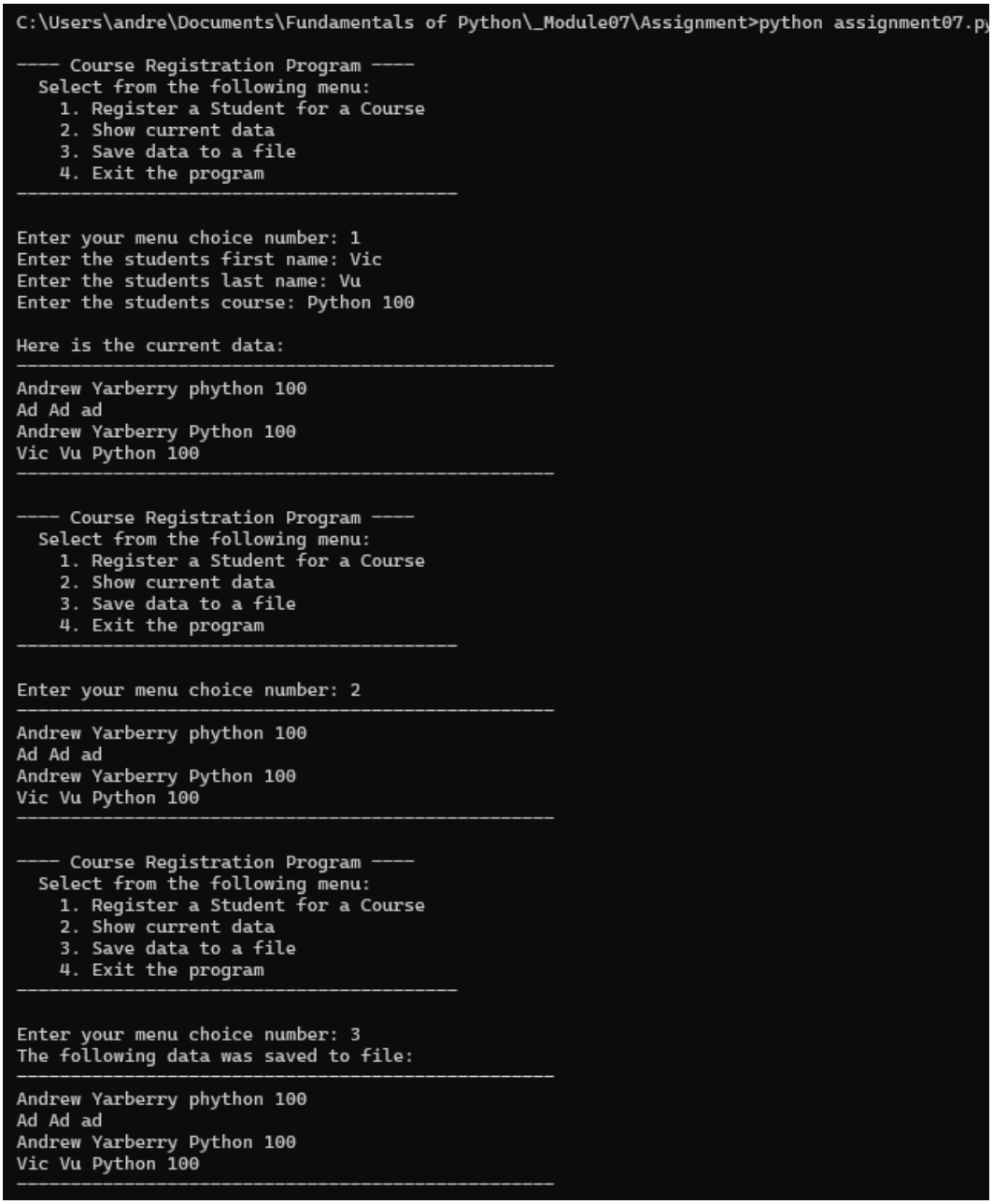

Figure 12: CMD Output

Summary

With this assignment we continue refactoring our code to further improve encapsulation by creating custom classes for a person and student. This converts input from the user into objects and that we can manipulate then convert back to string and store as a list of dictionaries before saving back to file.

References

1. *W3 Schools, https://www.w3schools.com/python/python_classes.asp, 2024 (External Site)*
2. *Real Python, https://realpython.com/python-getter-setter, 2024 (External Site)*
3. *Geeks for Geeks, https://www.geeksforgeeks.org/__init__-in-python/, 2024 (External Site)*
4. *Github, https://github.com/Finviel314/Python110, 2024 (External Site)*

Appendix

```
# ------------------------------------------------------------------------------
-------------- #
# Title: Assignment07
# Desc: This assignment further refactors are code by creating Person and
Student Classes
# Change Log: (Who, When, What)
#   #  Andrew Yarberry, 11/29/2024, Initial Release
# Notes:
#   * Add this starting data to the Enrollments.json file as needed.
# [{'FirstName': 'Vic', 'LastName': 'Vu', 'CourseName' : 'Python 100'
# ------------------------------------------------------------------------------
-------------- #

import json

# Constant Variables
FILE_NAME: str = 'Enrollments.json' # File name for storing student data
MENU = '''
---- 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
----------------------------------------
'''

#Main Data Structure
students: list = []  # Stores student information
menu_choice: str  # Holds the choice made by the user.
class Person:
    def __init__(self, first_name: str = '', last_name: str = ''):
        """
        Initializes a Person object.

        Args:
            first_name (str, optional): The person's first name. Defaults to
an empty string.
            last_name (str, optional): The person's last name. Defaults to an
empty string.
        """
        self.first_name = first_name
        self.last_name = last_name

    @property
    def first_name(self):
        """
```

```
        Gets the person's first name.

        Returns:
            str: The person's first name, capitalized.
        """
        return self._first_name.title()  # Capitalizes first letter of name

    @first_name.setter
    def first_name(self, value: str):
        """
        Sets the person's first name.

        Args:
            value (str): The new first name.

        Raises:
            ValueError: If the new first name contains non-alphabetic
characters.
        """
        if value.isalpha() or value == '':  # checks if alpha or number
            self._first_name = value
        else:
            raise ValueError("The last name should not contain numbers.")

    @property
    def last_name(self):
        """
        Gets the person's last name.

        Returns:
            str: The person's last name, capitalized.
        """
        return self._last_name.title()  # Capitalizes first letter of name

    @last_name.setter
    def last_name(self, value: str):
        """
        Sets the person's last name.

        Args:
            value (str): The new last name.

        Raises:
            ValueError: If the new last name contains non-alphabetic
characters.
        """
        if value.isalpha() or value == '':  # checks if alpha or number
            self._last_name = value
        else:
            raise ValueError("The last name should not contain numbers.")

class Student(Person):
    def __init__(self, first_name: str = '', last_name: str = '', course: str
= ''):
        """
        Initializes a Student object.
```

```
        Args:
            first_name (str, optional): The student's first name. Defaults to
an empty string.
            last_name (str, optional): The student's last name. Defaults to
an empty string.
            course (str, optional): The student's course. Defaults to an
empty string.
        """
        self._course = course
        super().__init__(first_name=first_name, last_name=last_name)
    @property
    def course(self):
        """
        Gets the student's course.

        Returns:
            str: The student's course.
        """
        return self._course

    @course.setter
    def course(self, value: str):
        """
        Sets the student's course.

        Args:
            value (str): The new course.
        """
        self._course = value

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

# File Processing Class
class FileProcessor:
    """
    Handles file operations for reading and writing student data to a JSON
file.

    Provides methods to:
    - Read student data from a specified JSON file.
    - Write student data to a specified JSON file.
    """

    @staticmethod
    def read_data_from_file(file_name: str, student_data):
        """
        Reads student data from a JSON file.

        Args:
            file_name (str): The name of the JSON file.
            student_data (list): A list of Student objects to populate.

        Returns:
            list: The updated list of Student objects.
        """
```

```
        try:
            with open(file_name, 'r') as file:
                list_of_dict = json.load(file)
                for student in list_of_dict:
                    student_object: Student =
Student(first_name=student["FirstName"],
                                            last_name=student['LastName'],
                                            course=student['CourseName'])
                    student_data.append(student_object)
        except FileNotFoundError as e:
            IO.output_error_messages('File does not exist, please create a
file called Enrollments.json.', e)
        except Exception as e:
            IO.output_error_messages('There was an non specific error.', e)
        return student_data

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """
        Writes student data to a JSON file.

        Args:
            file_name (str): The name of the JSON file.
            student_data (list): A list of Student objects to write.
        """
        try:
            list_of_dict: list = []
            for student in student_data:
                student_json: dict = {'FirstName': student.first_name,
                                      'LastName': student.last_name,
                                      'CourseName': student.course}
                list_of_dict.append(student_json)

            file = open(file_name, 'w')
            json.dump(list_of_dict, file, indent=4)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages('File does not exist, please create a
file called Enrollments.json.', e)
        except Exception as e:
            IO.output_error_messages('There was an non specific error.', e)

# End of Class Definition

# Input/Output Class
class IO:
    """
    A class for input/output operations in the course registration program.
    """
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """
        Prints an error message to the console.

        Args:
            message (str): The error message to display.
            error (Exception, optional): The exception object, if any.
```

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

    @staticmethod
    def output_student_courses(student_data: list):
        """
        Prints a formatted list of students and their courses.

        Args:
            student_data (list): A list of Student objects.
        """
        print("-" * 50)
        for student in student_data:
            print(student.first_name, student.last_name, student.course)
        print('-' * 50)

    @staticmethod
    def output_menu():
        """Prints the main menu to the console."""
        print(MENU)

    @staticmethod
    def input_menu_choice():
        """Prompts the user to enter a menu choice and validates the input.

        Returns:
            The user's selected menu choice as a string.
        """
        # Define function block
        choice = '0'
        try:
            choice = input('Enter your menu choice number: ')
            if choice not in ('1', '2', '3', '4'):
                raise Exception('You must choose 1, 2, 3, or 4')
        except Exception as e:
            IO.output_error_messages(e.__str__())
        return choice

    @staticmethod
    def input_student_data(student_data: list):
        """
        Prompts the user to enter student information and adds it to the
list.

        Args:
            student_data (list): A list of Student objects.
        """
        try:
            student = Student()
            while True: # Will check if alpha for name and loop until valid
entry
                try:
                    student.first_name = input('Enter the students first
name: ')
```

```
                    if not student.first_name.isalpha():
                        raise ValueError('Name cannot contain numbers or
special characters.')
                    break
                except ValueError as e:
                    print(e)
            while True:  # Will check if alpha for name and loop until valid
entry
                try:
                    student.last_name = input('Enter the students last name:
')
                    if not student.last_name.isalpha():
                        raise ValueError('Name cannot contain numbers or
special characters.')
                    break
                except ValueError as e:
                    print(e)

            student.course = input('Enter the students course: ')
            student_data.append(student)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
# Code for that does not loop and ask for continued input until correct
format (can delete later)
        # try:
        #     # Input Data
        #         student= Student()
        #         student.first_name = input('Enter the students first name:
')
        #         student.last_name = input('Enter the students last name: ')
        #         student.course = input('Enter the students course: ')
        #         student_data.append(student)
        # except ValueError as e:
        #     IO.output_error_messages("That value is not the correct type of
data!", e)
        # except Exception as e:
        #     IO.output_error_messages("There was a non-specific error!", e)
        # return student_data

# Start of Main

students = FileProcessor.read_data_from_file(file_name=FILE_NAME,
student_data=students)

while True:
    IO.output_menu()
    menu_choice = IO.input_menu_choice()

    if menu_choice == '1':
        IO.input_student_data(student_data=students)
        print('\nHere is the current data:')
        IO.output_student_courses(student_data=students)

    elif menu_choice == '2':
        IO.output_student_courses(student_data=students)

    elif menu_choice == '3':
```

```
        FileProcessor.write_data_to_file(FILE_NAME, student_data=students)
        print('The following data was saved to file: ')
        IO.output_student_courses(student_data=students)

    elif menu_choice == '4':
        print('Goodbye!')
        break  # out of the while loop
```

Figure 13: Full Python Code