Name: Parth Dave

Date: August 27, 2025

Course: IT FDN 110 A Su 25: Foundation of Programming: Python

Assignment: Assignment07

Introduction

This week, I explored advanced concepts of Object-Oriented Programming (OOP) in Python. I reviewed how programs use statements, functions, and classes, and how classes enable reusable and modular code. A class serves as a blueprint for objects, defining their structure and behavior through attributes (data/state) and methods (functions). Classes support encapsulation, abstraction, and reusability, which are essential for modular programming.

I learned about constructors, special methods defined with __init__() that automatically initialize an object's attributes when an instance is created. Attributes belong to a class and represent the data or state of an object, typically defined using the self keyword. I also studied getter and setter properties, which control access to attributes, while the @property decorator allows data validation when getting or setting values.

Additionally, I explored class inheritance, where a child class inherits properties and methods from a parent class. Inheritance reduces code duplication, improves maintainability, and provides flexibility to override parent methods. The super() function is particularly useful for calling a parent class constructor from a child class. Finally, I learned how magic methods like __init__() and __str__() work and how to convert objects into dictionaries for storage, bridging the gap between object-oriented structures and data handling.

Description

This Python script implements a Course Registration Program using object-oriented programming, data validation, and structured error handling. It defines two classes, Person and Student, where Student inherits from Person. Both classes use private attributes with getter and setter properties to validate input, and the __str__() method is overridden to display object data in a readable format.

Following the Separation of Concerns (SoC) principle, two main classes were implemented:

- FileProcessor Class
    - Handles reading and writing student data to a JSON file. It converts JSON dictionaries into Student objects when reading and vice versa when writing.
    - Includes appropriate error-handling methods to ensure reliability and robustness.
- IO Class
    - Contains multiple functions with local variables for safer execution.

- Provides functionality for printing menus, capturing user input, registering student data, and displaying stored records.

To improve maintainability and readability, docstrings were added to each class and function. These inline documentation blocks enhance the reliability of the code by making its purpose and behavior clear for future use and collaboration.

Code Snippet:

Code repository on GitHub: https://github.com/ppdave19/IntroToProg-Python-Mod07

```
#
------------------------------------------------------------------------------
------------ #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
#   Parth Dave ,08/27/2025,Created Script
#
------------------------------------------------------------------------------
------------ #
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"

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


# TODO Create a Person Class
class Person:
# TODO Add first_name and last_name properties to the constructor
   def __init__(self, first_name: str = "", last_name: str = ""):
       self.first_name = first_name
       self.last_name = last_name

# TODO Create a getter and setter for the last_name property
   @property
   def first_name(self) -> str:
```

```
        return self.__first_name

    @first_name.setter
    def first_name(self, value: str):
        if not value.strip().isalpha():
            raise ValueError("First name must only contain letters.")
        self.__first_name = value.strip().title()

# TODO Create a getter and setter for the first_name property
    @property
    def last_name(self) -> str:
        return self.__last_name

    @last_name.setter
    def last_name(self, value: str):
        if not value.strip().isalpha():
            raise ValueError("Last name must only contain letters.")
        self.__last_name = value.strip().title()

# TODO Override the __str__() method to return Person data
    def __str__(self):
        return f"{self.first_name},{self.last_name}"

# TODO Create a Student class the inherits from the Person class
class Student(Person):

# TODO call to the Person constructor and pass it the first_name and last_name
data
# TODO add a assignment to the course_name property using the course_name
parameter
    def __init__(self, first_name: str = "", last_name: str = "", course_name:
str = ""):
        super().__init__(first_name, last_name)
        self.course_name = course_name

# TODO add the getter for course_name
    @property
    def course_name(self) -> str:
        return self.__course_name

# TODO add the setter for course_name
    @course_name.setter
    def course_name(self, value: str):
        if not value.strip():
            raise ValueError("Course name cannot be empty.")
        self.__course_name = value.strip().title()

# TODO Override the __str__() method to return the Student data
    def __str__(self):
        return f"{self.first_name},{self.last_name},{self.course_name}"

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

```
    ChangeLog: (Who, When, What)
    Parth Dave ,08/27/2025,Created Class
    """
    @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
        then returns the list filled with student data.

        ChangeLog: (Who, When, What)
        Parth Dave ,08/27/2025,Created function

        :param file_name: string data with name of file to read from

        :return: list
        """
        try:
            # Get a list of dictionary rows from the data file
               # TODO replace this line of code to convert dictionary data to
Student data
            file = open(file_name, "r")
            json_students = json.load(file)

            for item in json_students:
                      student = Student(item['first_name'], item['last_name'],
item['course_name'])
                student_data.append(student)

        except Exception as e:
                     IO.output_error_messages("Error: There was a problem with
reading the file.", e)

        except FileNotFoundError as e:
             IO.output_error_messages("Text file must exist before running this
script!",e)

        except Exception as e:
             IO.output_error_messages(message="Error: There was a problem with
reading the file.", error=e)

        return student_data

    @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)
        Parth Dave ,08/27/2025,Created Function

        :param file_name: string data with name of file to write to
        :param student_data: list of dictionary rows to be writen to the file

        :return: None
        """
```

```
        student_objects = []
        try:
            # TODO Add code to convert Student objects into dictionaries (Done)

            file = open(file_name, "w")
            data = []
            for student in student_data:
                student_dict = {
                    'first_name': student.first_name,
                    'last_name':student.last_name,
                    'course_name':student.course_name
                }
                data.append(student_dict)
            json.dump(data, file)
            IO.output_student_and_course_names(student_data=student_data)
            file.close()

        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 file.closed == False:
                file.close()


# Presentation ------------------------------------- #
class IO:
    """
     A collection of presentation layer functions that manage user input and
output

    ChangeLog: (Who, When, What)
    Parth Dave ,08/27/2025,Created Class
    Parth Dave ,08/27/2025,Added menu output and input functions
    Parth Dave ,08/27/2025,Added a function to display the data
    Parth Dave ,08/27/2025,Added a function to display custom error messages
    """

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

        ChangeLog: (Who, When, What)
        Parth Dave ,08/27/202, 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')
```

```
    @staticmethod
    def output_menu(menu: str):
        """ This function displays the menu of choices to the user

        ChangeLog: (Who, When, What)
        Parth Dave ,08/27/2025, Created Function


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

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

        ChangeLog: (Who, When, What)
        Parth Dave ,08/27/2025, 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

    @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)
        Parth Dave ,08/27/2025, Created Function

        :param student_data: list of dictionary rows to be displayed

        :return: None
        """

        print("-" * 50)
        for student in student_data:
            # TODO Add code to access Student object data instead of dictionary
data
                print(f'Student {student.first_name} {student.last_name} is
enrolled in {student.course_name}')
        print("-" * 50)

    @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)
       Parth Dave ,08/27/2025, Created Function

         :param student_data: list of dictionary rows to be filled with input
data

       :return: list
       """

       try:
           student_first_name = input("Enter the student's first name: ")
           if not student_first_name.isalpha():
               raise ValueError("The last name should not contain numbers.")
           student_last_name = input("Enter the student's last name: ")
           if not student_last_name.isalpha():
               raise ValueError("The last name should not contain numbers.")
           course_name = input("Please enter the name of the course: ")

              # TODO Replace this code to use a Student objects instead of a
dictionary objects
           student=Student(student_first_name, student_last_name,course_name)
           student_data.append(student)
           print()
                           print(f"You  have  registered  {student_first_name}
{student_last_name} for {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


# 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_and_course_names(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
    else:
        print("Please only choose option 1, 2, or 3")

print("Program Ended")
```

Sample Output

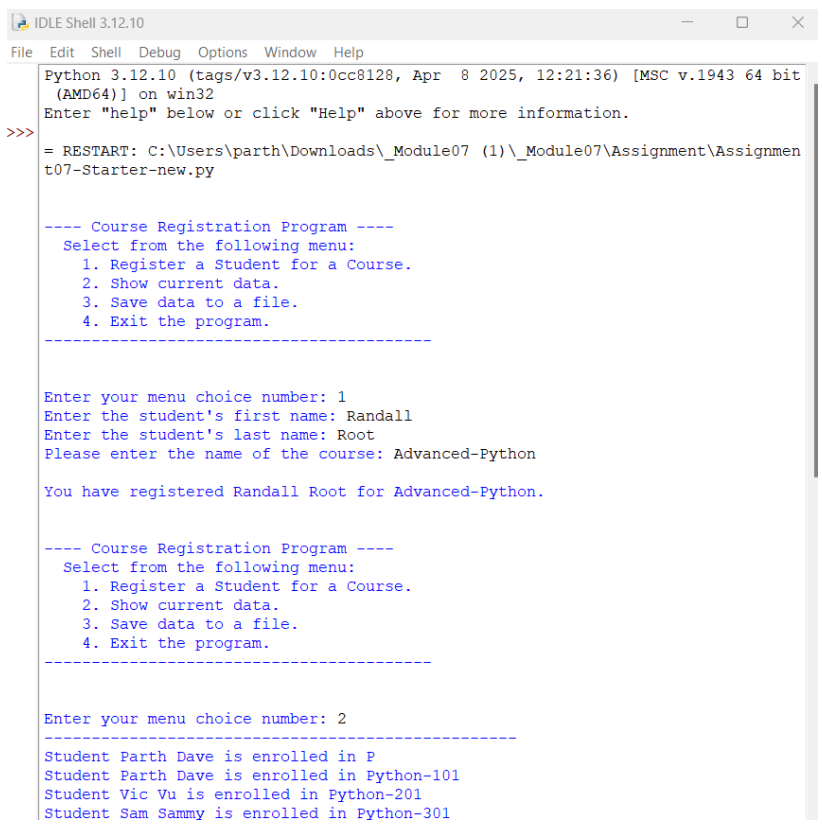

Figure 1: Python IDLE Terminal Output

```
-------------------------------------------------
Student Parth Dave is enrolled in P
Student Parth Dave is enrolled in Python-101
Student Vic Vu is enrolled in Python-201
Student Sam Sammy is enrolled in Python-301
Student Randall Root is enrolled in Advanced-Python
-------------------------------------------------


---- 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 Parth Dave is enrolled in P
Student Parth Dave is enrolled in Python-101
Student Vic Vu is enrolled in Python-201
Student Sam Sammy is enrolled in Python-301
Student Randall Root is enrolled in Advanced-Python
-------------------------------------------------


---- 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 2 : Python IDLE Terminal Output

```
C:\Users\parth\AppData\Local\Programs\Python\Python312\python.exe "C:\Users\parth\Downloads\_Module07 (1)\_Module07\Assignment\Assignment07-Starter-new.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.
----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: Dan
Enter the student's last name: Danny
Please enter the name of the course: C#-101

You have registered Dan Danny for C#-101.


---- 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 3 : PyCharm Terminal Output

Summary

This week, I learned that in Python OOP, a class is a blueprint for objects that encapsulates data and behavior. Classes use constructors, properties, inheritance, `super()`, and magic methods like `__str__()` to create reusable, modular, and well-structured code.