Pattama Srianomai

Aug 26, 2025

Foundations of Programming: Python

Assignment 07
https://github.com/patanomai/IntroToProg-Python-Mod07

Steps to complete Assignment 07:

Data Processing Using Classes, Class Properties, Constructors, The Self Keyword, Inheritance, and Overriding Methods

Introduction

In this paper, I will describe the steps I took to create a Python Script using classes, class properties, constructors, the self-keyword, inheritance, and overriding methods. Guided by the notes and videos provided in Module 07. I will walk through each step, beginning with opening and reviewing the starter file Assignment07-Starter.py.

Step-by-Step Process

Step 1. Opening and reviewing the starter file Assignment07-Starter.py

Opening a file in PyCharm

1. Use the Project Pane on the left side
2. Double-click the Assignment 07-Starter.py
3. It will appear in the editor window

Renaming a file in PyCharm

1. Right-click the Assignment 07-Starter.py on the Project Pane
2. Select Rename
3. Rename the file to Assignment06.py
4. Click Refactor

Step 2. Updating a script header

Update the script header with my name, current date, and change history. *My script header is shown in Figure 1 below.*

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
# PSrianomai,8/23/2025,Created Script
# ------------------------------------------------------------------------------------------ #
```

Figure 1: Script Header

Step 3. Defining Data Constants and Global Variables

Assignment 07 - Task 4 outlined the required constants and variables for this exercise:

Constants:

- The constant **MENU: str** is set to the value:

  ---- 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 constant **FILE_NAME: str** is set to the value "Enrollments.json"
- Constant values do not change throughout the program.

Variables:

- **menu_choice: str** is set to empty string.
- **students: list** : list is set to and empty list

Figure 2 presents the constants and global variables used in my script.

```python
import io as _io #needed to try closing in the finally block
import json #import code from Python's JSON module into my script

# 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.
```

Figure 2: Constants and Global Variables definition

Step 4. Processing and presenting the data

This assignment builds on the structure introduced in Assignment 06, but it adds set of data classes. Task 4 outlined the requirements of Classes, Class Properties, Class Methods, functions, Input/Output, Processing, and Error Handling.

Classes:

- The program includes a class named FileProcessor.
- The program includes a class named IO.
- The program includes a class named Person.
- The program includes a class named Student.
- All classes include descriptive document strings.

Class Properties:

- The program includes properties for **student_first_name: str** and defaults to an empty string.
- The program includes properties for **student_last_name: str** and defaults to an empty string.
- The program includes properties for **course_name: str** and defaults to an empty string.
- The program's properties must include simple validation code.

Class Methods:

- The program includes a method to extract comma separately data from each data class.

Functions:

- All functions include descriptive document strings.
- All functions the include exception blocks use the output_error_messages() function for handling error messages.
- All non-instance functions use the @staticmethod decorator (The ones in the FileProcessor and IO classes.)
- The program includes functions with the following names and parameters:
  - output_error_messages(message: str, error: Exception = None)
  - output_menu(menu: str)
  - input_menu_choice()
  - output_student_courses(student_data: list)
  - input_student_data(student_data: list)
  - read_data_from_file(file_name: str, student_data: list):
  - write_data_to_file(file_name: str, student_data: list):

Processing:

- When the program starts, the contents of the "Enrollments.json" are automatically read into a two-dimensional list of dictionaries rows using the json.load() function. Next, it converts that data into a list of Student object rows. (**Tip:** Make sure to put some starting data into the file or you will get an error!)
- On menu choice 3, the program opens a file named "Enrollments.json" in write mode using the open() function. Next, it converts the data in the **students** variable (a list of Student object rows) into a list of dictionary rows, then writes the list of dictionary data into the file using the json.dump() function. Finally, it closes the file using the close() method.
- On menu choice 4, the program ends.

Error Handling:

- The program provides structured error handling when the file is read into the list of dictionary rows.
- The program provides structured error handling when the user enters a first name.
- The program provides structured error handling when the user enters a last name.
- The program provides structured error handling when the dictionary rows are written to the file.

To meet these requirements, I separated the script into data, processing, and presentation layers.

Figure 3 - Data: class Person, class Student, which includes

- *first_name and last_name properties to the constructor*
- *Getter and setter for the first_name and last_name properties*
- *Override the the __str__() method to return Person data*
- *Inherits the Student class from the Person class*

```python
# TODO Create a Person Class (Done)
class Person: # no need to specify object. Python automatically do it  1 usage
    """
    A class representing person data.

    Properties:
        student_first_name(str): The student's first name.
        student_last_name(str): The student's last name.

    ChangeLog:
        - PSrianomai, 8.23.2025: Created the class.
    """

# TODO Add first_name and last_name properties to the constructor (Done)
    #Per Assignment Class Properties section: The program includes properties
    #for student_first_name: str and defaults to an empty string
    #The program includes properties for student_last_name: str and
    #defaults to an empty string
    #constructor with private attributes __init__
    def __init__(self, student_first_name:str = '', student_last_name: str = ''):
        self.student_first_name = student_first_name
        self.student_last_name = student_last_name

# TODO Create a getter and setter for the first_name property (Done)
    @property # (Use this decorator for the getter or accessor)
    def student_first_name(self):
        return self.__student_first_name.title() # formatting code

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

```python
# TODO Create a getter and setter for the last_name property (Done)
    @property
    def student_last_name(self):
        return self.__student_last_name.title() # formatting code

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

# TODO Override the __str__() method to return Person data (Done)
    # return a coma-separated string of data
    def __str__(self):
        return f'{self.student_first_name},{self.student_last_name}'

# TODO Create a Student class the inherits from the Person class (Done)
class Student(Person): # we call Student a child or sub class,
# and we call Person a parent or super class
    """
    A class representing student data.

    Properties:
        student_first_name (str): The student's first name.
        student_last_name (str): The student's last name.
        course_name (str): The student's course name.

    Changelog: (Who, When, What)
    PSrianomai, 8.23.2025, Create Class
    PSrianomai, 8.23.2025, Added properties and private attributes
    """
```

```python
# TODO call to the Person constructor and pass it the first_name and (Done)
# last_name data
    #defines a constructor that accepts three parameters
    def __init__(self, student_first_name: str = '',
                 student_last_name: str = '',
                 course_name: str = ''):
        #use super method to call the constructor of the person class
        super().__init__(student_first_name=student_first_name,
                         student_last_name=student_last_name)

# TODO add a assignment to the course_name property using the
# course_name parameter (Done)
        self.course_name = course_name

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

# TODO add the setter for course_name (Done)
    @course_name.setter
    def course_name(self, value: str): # I removed validation to
                                       # allow input as Python 100 format
        #if value.isalpha() or value == "":  # is character or empty string
        self.__course_name = value
        #else:
            #raise ValueError("The course name should not contain numbers.")

# TODO Override the __str__() method to return the Student data (Done)
    def __str__(self):
        return f'{self.student_first_name},{self.student_last_name},{self.course_name}'
```

Figure 3: Data: class Person, class Student

I followed the approach from Mod07-Lab03, creating constructors with private attributes, property getter and setter, inheritance, and overriding methods.

Figure 4 - Processing: class FileProcessor, which includes two functions

- *read_data_from_file () uses json.load() function to load data into a list of dictionary*
- *write_data_to_file () writes the student data to Enrollments.json using json.dump()*

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

    ChangeLog: (Who, When, What)
    PSrianomai,8.23.2025,Created Class
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ Reads data from a JSON file and appends Student objects to
        the provided list. Also returns a separate list of Student objects.

        ChangeLog: (Who, When, What)
        PSrianomai,8.23.2025,Created function
        PSrianomai,8.23.2025, Converted list of dictionaries to
        list of student objects

        :param file_name: string data with name of file to read from
        :param student_data: list of dictionary rows to be filled with
        file data

        :return: list of Student objects (also appends to student_data list)
        """
        # Convert the list of dictionary rows into a list of Student objects
        # student_objects = [] --want to move to outside try block as Pycharm
        # flag error because if exception occurs before student_objects is
        # defined, then Python won't know what to return but it doesn't meet
        # what listed in the Processing section in the Assignment.
        file = _io.TextIOWrapper  # add file as a local variable
        try:
            # Get a list of dictionary rows from the data file
            file = open(file_name, "r")
            json_students = json.load(file) #load function returns a list of
                                            # JSON dictionary row

            # Convert the list of dictionary rows into a list of Student objects
            student_objects = []
```

```python
        # TODO replace this line of code to convert dictionary data
        # to Student data (Done)
        #student_objects = json_students
        for student in json_students:
            student_object: Student = Student(
                                student_first_name=student["FirstName"],
                                student_last_name=student["LastName"],
                                course_name=student["CourseName"])
            student_data.append(student_object) #convert dictionary data to
                                                # Student data
            #student_objects.append(student_object) #append to a
            #Student objects

        file.close()

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

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

    return student_data
```

```python
    @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)
        PSrianomai,8.23.2025,Created function

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

        :return: None
        """
        file = _io.TextIOWrapper  # add file as a local variable
        try:
            file = open(file_name, "w")
            # TODO Add code to convert Student objects into dictionaries (Done)
            # it was listed as Done by Prof.
            # Create a new list to hold Json data to use with the json.dump()
            # function.
            list_of_dictionary_data: list = []
            for student in student_data:  # Convert List of Student objects
                                          # to list of dictionary rows.
                student_json: dict \
                    = {"FirstName": student.student_first_name,
                       "LastName": student.student_last_name,
                       "CourseName": student.course_name}
                list_of_dictionary_data.append(student_json)
            #file = open(file_name, "w")
            # Change the first argument to be the list of dictionary data
            json.dump(list_of_dictionary_data, file)
            #json.dump(student_data, file)
            file.close()
            IO.output_student_and_course_names(student_data=student_data)
        except Exception as e:
            message = "Error: There was a problem with writing to the file.\n"
            message += ("Please check that the file is not open by "
                        "another program.")
            IO.output_error_messages(message=message,error=e)
        finally:
            if file.closed == False:
                file.close()
```

Figure 4: Processing: class FileProcessor

Input / Output:

- On menu choice 1, the program prompts the user to enter the student's first name and last name, followed by the course name, using the input() function and stores the inputs in the respective variables.
- Data collected for menu choice 1 is added to the **students** two-dimensional list of Student objects.
- On menu choice 2, the program uses the print() function to show a string of comma-separated values for each row collected in the **students** variable.

- On menu choice 4, the program ends.

Figure 5 shows my script for IO class in the presentation layer, which incorporates all input and output functions:

- *output_error_messages (message: str, error: Exception = None)*
- *output_menu (menu: str)*
- *input_menu_choice ()*
- *output_student_courses (student_data: list)*
- *input_student_data (student_data: list)*

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

    ChangeLog: (Who, When, What)
    PSrianomai,8.23.2025,Created Class, Added menu output and input
    functions, Added a function to display the data, 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)
        PSrianomai,8.23.2025,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)
        PSrianomai,8.23.2025,Created function

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

```python
@staticmethod  1 usage
def input_menu_choice() -> str:
    """ This function gets a menu choice from the user

    ChangeLog: (Who, When, What)
    PSrianomai,8.23.2025,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  2 usages
def output_student_and_course_names(student_data: list):
    """ This function displays the student and course names to the user

    ChangeLog: (Who, When, What)
    PSrianomai,8.23.2025,Created function
    PSrianomai,8.24.2025,Converted code to use student objects
    instead of dictionaries

    :param student_data: list of student object data to be displayed

    :return: None
    """

    print("-" * 50)
    for student in student_data:
```

```python
        # TODO Add code to access Student object data instead of
        # dictionary data (Done)
        print(student.student_first_name,student.student_last_name,
              student.course_name, sep=",") # separate by comma
    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)
    PSrianomai,8.23.2025,Created function
    PSrianomai,8.24.2030,Converted code to use student objects
    instead of dictionaries

    :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 first 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 (Done)
        student = Student(student_first_name=student_first_name,
                          student_last_name=student_last_name,
                          course_name=course_name)
        student_data.append(student)
        print()
        print(f"You have registered {student_first_name} "
```

```
                f"{student_last_name} "
                f"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
#  End of function definitions
```

Figure 5: IO class with all input and output functions

Figure 6 presents the main body of the script, which incorporates a for loop statement with call functions.

```
# 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")
```

Figure 6: Incorporate a for loop to loop and call functions.

Step 5. Testing the Script

I tested the script in PyCharm and the Command Prompt. *The results are shown in Figure 7.* After running the script, I verified that the JSON file was saved successfully on my computer *(see Figure 8).*

PyCharm:

```
C:\Python\Python3.13\python.exe C:\Users\taeya\Documents\Python\PythonCourse\A07\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.
----------------------------------------


Enter your menu choice number: 2
-------------------------------------------------
Vic,Vu,Python
Sue,Jones,Python
Pat,Srianomai,Python
Am,Serm,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: 1
Enter the student's first name: Ty
Enter the student's last name: Vu
Please enter the name of the course: Python

You have registered Ty Vu for 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
--------------------------------------------------
Vic,Vu,Python
Sue,Jones,Python
Pat,Srianomai,Python
Am,Serm,Python
Ty,Vu,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

Process finished with exit code 0
```

Command Prompt:

```
C:\Users\taeya\Documents\Python\PythonCourse\A07>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.
----------------------------------------


Enter your menu choice number: 2
-------------------------------------------------
Vic,Vu,Python
Sue,Jones,Python
Pat,Srianomai,Python
Am,Serm,Python
Ty,Vu,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: 1
Enter the student's first name: Tai
Enter the student's last name: Vu
Please enter the name of the course: Python

You have registered Tai Vu for 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
--------------------------------------------------
Vic,Vu,Python
Sue,Jones,Python
Pat,Srianomai,Python
Am,Serm,Python
Ty,Vu,Python
Tai,Vu,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 6. Script results

Figure 8. Saved JSON file

Summary

This assignment demonstrates how object-oriented programming principles such as class design, inheritance, and method overriding can be applied to build a structured and user-friendly Python script. I relied heavily on the lab materials for guidance, especially when certain concepts felt confusing.