Ronald Mursewick
November 27, 2023
Foundations of Programming:
Python Assignment 07
https://github.com/Visual-Library/IntroToProg-Python-Mod07

Classes, Constructors, Properties, and Inheritance

Introduction

The purpose of this writing is to document some enhancements to our prior program. In this iteration of our program we will add Classes, constructors, properties and inheritance. Classes allow us to logically associate types of data. For example we might have a Class called “Vehicles”. For some subclasses to Vehicles we might have cars, trucks, planes, and motorcycles. One common subclass to cars and trucks might be “Volvo” since they manufacture cars and trucks. Constructors allow us to “form” the data. Inheritance allows us to inherit the methods and properties from another class.

Classes

Our parent class in this case will be person. Two of the details that will be common to all of the students, teachers or faculty will be the persons first and last name.

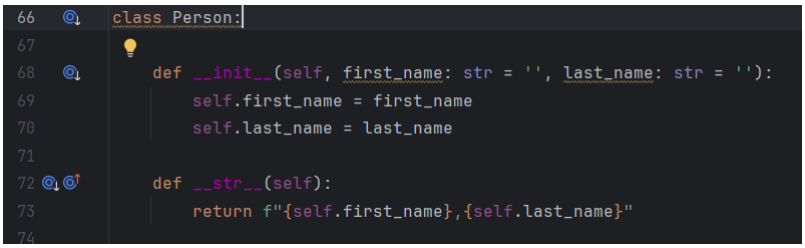

Figure 1. Class definition (Parent Class)

Next we will define the constructors. The purpose of the constructor is to create and instance (instantiate) the. This is performed with the __init__ function.

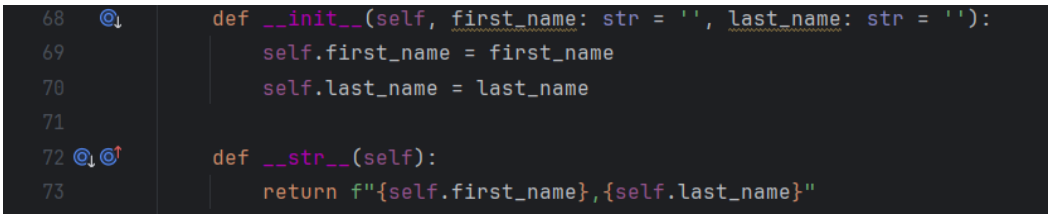

Figure 2. Constructors details

Inheritance is the process of passing methods and properties. In our use a “Student” is a subset of the “Person” class. So the student inherits the first name and last name from the “Person” or parent class then adds three other values that are unique to the student. The values are not applicable to “Teacher” or “Faculty”. Inheritance is accomplished in Python with the use of the “super” method.

```
class Student(Person):  # Add a class name to indicate explicit inheritance.

    def __init__(self, first_name: str = '', last_name: str = '', class_name: str = ''):
        super().__init__(first_name=first_name, last_name=last_name)

        self.course_name = course_name
        self.course_price = COURSE_PRICE
        self.course_cost = TOTAL_PRICE

    def __str__(self):
        return f"{self.first_name},{self.last_name},{self.course_name},{self.course_price},{self.course_cost}"
```

Figure 3. Inheritance (Child Class)

Pandas

```
# import pandas to help use 2 dimensional arrays easily
import pandas as pd

#
# Create the dataframe with the header row

students_pd = pd.DataFrame(columns=("FirstName", "LastName", "CourseName", "CoursePrice", "CourseCost"))
```

Figure 4. Importing Pandas methods and creating the student data DataFrame

Our DataFrame will be 5 columns wide and able to have rows added indefinitely. In the DataFrame creation above we are creating the header row for the data.

```
    #
    # we will append the newly added student to the dataframe
    # this DataFrame can extend indefinitly

    students_pd.loc[len(students_pd)] = student_data
```

Figure 5. Appending the newly added student to the DataFrame

One of the changes is using pandas over variables. Notice the “0” just to the left of the last name “Harrison”. When the operator selects option “2”, we print the DataFrame. With indexing the “zero” row is the first row of the array. Since Harrison was our first student entered he is saves in the table in row zero

```
Total Students registered so far is:  1

  FirstName LastName  CourseName  CoursePrice  CourseCost
0  Harrison     Ford  Python 100       999.98     1089.98
```

Figure 6. Shows how data is stored in the DataFrame

Moving the student data from the DataFrame we build with the information the operator entered during the program run. Another reason for using pandas is ease of data handling. Here we see that with a one line command (method) we are able to write the entire contents of the DataFrame to the .json file. We choose the mode=”a” so we append our new data to any other data that has been entered into the file. We will also use the orient=”records” to keep our data in a tabular format.

```
173         #
174         # now we will append the newly added students to the .json file
175
176         students_pd.to_json(FILE_NAME, mode="a", index=False, lines=True, orient="records")
177
```

Figure 7. Use of the DataFrame to .json file method.

In our program we will also show the use of the list and dictionary operations with output to a second .json file.

```
51
52  # this is a list in dictionary format
53  student_list        : list[dict[str]] = []
54
```

Figure 8. List creation utilizing dictionary format

```
181
182         #
183         # append the current student data to the list
184
185         student_list.append(student_data)
186
```

Figure 9. Appending the data just entered to our running list.

Finally in option 3 we will create or append our data to a second .json file. Note the format varies slightly. We will use the json dump method to perform this operation.

```
234         #
235         # we will also write out the student list file we created as well
236
237         file_obj = open("Enrollments2.json", "a+")
238         json.dump(student_list, file_obj)
239         file_obj.close()
```

Figure 10. Output to file using json dump.

Another process introduced in this lesson is validating data with the try-raise-except. The way we use the check in out code is to verify that the first and last names are alphabetical. We use the .isalpha method to test the operators input to confirm that it is a name.

```
    #
    # confirm the student first name is alphebetic other wise go back and ask again

    try:
        if not student_first_name.isalpha():
            raise ValueError("The first name must be alphabetic. ")
    except ValueError as e:
            print("\n", e, "\n")
            continue
```

Figure 11. using try-raise-except example

Program Operation

First we will choose operation 1 to register a student.

```
C:\Users\Ron\PycharmProjects\pythonProject>python main.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 Selection:
1

Enter the students first name:
neil
Enter the students last name:
armstrong
```

Figure 12. Menu selection 1

Now the operator chooses select 2, so we show the operator what data has been entered.

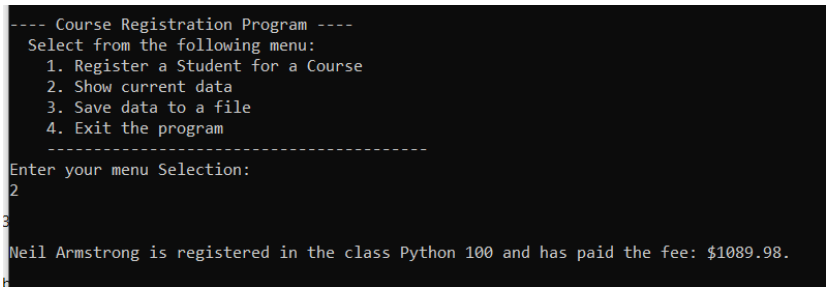

Figure 13. Menu selection 2

In menu select 3, we save the data to the .csv file. In this case another operator has entered two other students. In the beginning of the program we checked to see if a student data file existed. If the file existed we opened the for "append" to we could add data to the already registered student list. After we write the latest student to the file we show all of the students that have been registered for the class. Another change in the output for selection 3, was to change $stdout to a file and print the enrollment list to a file. Once the output is complete change $stdout back to the console for dialogue to the operator.

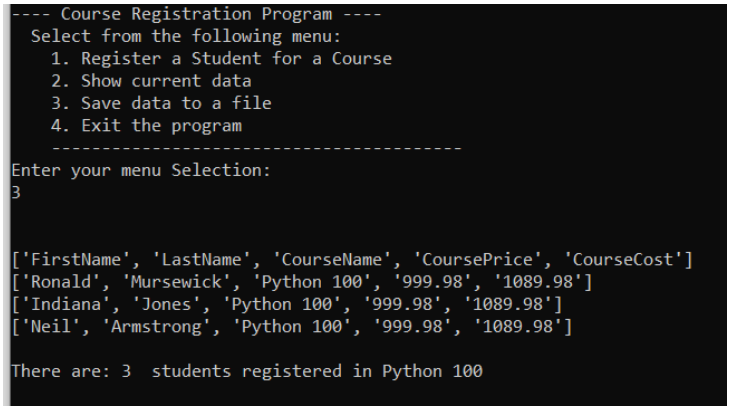

Figure 14. Menu selection 3

Menu selection ends the program and returns us to the command prompt.

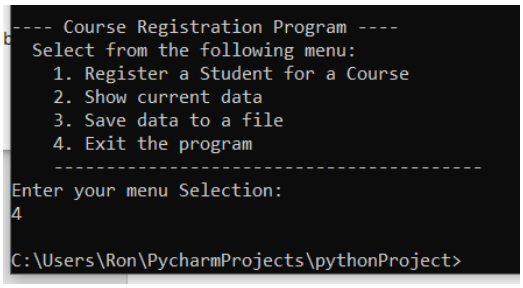

Figure 9. Menu selection 15

We put some error processing code in so that if the operator tried to show the current data before a student was registered, we would let them know that they needed to enter student data.

```
---- 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 Selection:
2


No students have been registered yet
```

Figure 16. Menu selection 2, when no student has been registered.

The Enrollment.json data file as displayed in Notepad.

Enrollments.json - Notepad

File Edit Format View Help


```
{"FirstName":"Ronald","LastName":"Mursewick","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Indiana","LastName":"Jones","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Alan","LastName":"Shepard","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Neil","LastName":"Armstrong","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Marion","LastName":"Ravenswood","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Reunion","LastName":"Band","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Reunion","LastName":"Band","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Maria","LastName":"Seiler","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Abraham","LastName":"Lincoln","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"George","LastName":"Washington","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Herbert","LastName":"Hoover","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"John","LastName":"Hancock","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"James","LastName":"Madison","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Alexander","LastName":"Hamilton","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Jimmy","LastName":"Carter","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
{"FirstName":"Neil","LastName":"Diamond","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
```


Figure 17. Enrollments.json data file

Enrollments2.json - Notepad

File Edit Format View Help


```
[{"FirstName": "Gordon", "LastName": "Cooper", "CourseName": "Python 100", "CoursePrice": 999.98, "CourseCost": 1089.98}][{"FirstName": "Frank", "LastName"
```


Figure 18. The Enrollments2.json data file

Assignment06.txt - Notepad

File Edit Format View Help


```
Student:  {"FirstName":"Elton","LastName":"John","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
Student:  {"FirstName":"John","LastName":"Glenn","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
Student:  {"FirstName":"Alan","LastName":"Shepard","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
Student:  {"FirstName":"Gus","LastName":"Grissom","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
Student:  {"FirstName":"Ed","LastName":"White","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
Student:  {"FirstName":"Jim","LastName":"Lovell","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
Student:  {"FirstName":"Roger","LastName":"Chaffee","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
Student:  {"FirstName":"Gordon","LastName":"Cooper","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
Student:  {"FirstName":"Frank","LastName":"Borman","CourseName":"Python 100","CoursePrice":999.98,"CourseCost":1089.98}
Total students registered for the class is: 9
```


Figure 19 Reassigning $stdout to a .txt file for printing

Summary

We continue to build upon our first assignment. As complexity grows we want to be able to isolate or encapsulate data. We accomplish this by adding classes with constructors. The purpose of the is so the various methods will not corrupt the data of another method. Now we add operations to perform repetitive tasks until a specified condition is met. We are creating function for repetitive procedures in our code. The function be used in a variety of forms: Parameter(s) requires, no parameters, data returned, only output. With adding complexity we want to look for methods that enhance our ability to manage larger amounts of data as efficiently as possible. We show how data can be used in lists with dictionary and Pandas is a powerful tool to aid us with data management and manipulation. We change out output file type to json since it is widely is for data manipulation today. We also add data checking to ensure that the data we write to out file is clean or normalized to minimize the chance for errors or misinterpretation of data later in the processing.