

Jessica Boardman

August 14, 2024

IT FND 110

Assignment 07

GitHub: <https://github.com/jboarduw/IntroToProg-Python-Mod07>

# Data Classes & Objects

## Introduction

This week I learned how to create data classes and use instances of a class, objects, in a Python script. The script I created has two data classes, one for person data and one for student data. In each class, constructors are used to initialize the attributes of an object and properties manage those attributes. Upon completion of authoring the program, I was able to execute the script in both PyCharm and the MacOS terminal. The steps I took to complete Assignment 07 are expanded below.

## Creating the Program

Before attempting to create the program, I read through the entire acceptance criteria in the assignment to gain a holistic view of what the program needed to do from beginning to end. I authored the Python program using the starter file for Assignment07.py in PyCharm.

## Objects

At the core of this week's learning was an understanding of how object oriented programming languages like Python use Objects to represent a specific instance of a class. In this situation,

the class serves as the template and the object is the copy of that template which is then modified. The value gained from creating an object from a class is:

- Isolation - changes made to the object data do not affect any other objects of the class.
- Reusability - the class is a template that can be used to make as many objects as needed without repeating code.
- Abstraction - simplifies the complexity of code by enabling interaction with objects absent concerns on the data being stored.

## Data Classes & Constructors

In the reading this week, we learned about data classes, the templates from which we create objects, and wrote a Python script that leverages two classes. The Person class has attributes of a first name and last name (Figure 1.1). After defining the class, there is a constructor that sets the object's attributes when it is first created. The constructor is shown as `__init__` and will set the first name and last name to a blank string when the object is created (Figure 1.1).

```
29  @ class Person: # (jb 8.12.24) person data class
30      """
31      A collection of person data
32      ChangeLog: (Who, When, What)
33      JBoardman,8.12.2024, Created Class
34      """
35
36      # (jb 8.12.24) added first_name and last_name properties to the constructor
37      # (jb 8.12.2024) intentionally not using 'student' here as this is a person and later could be a non-student
38  @ def __init__(self, first_name: str = '', last_name: str = ''): # (jb 8.12.24) created constructor
39      """
40      Initialize object and set first name and last name to blank string
41      ChangeLog: (Who, When, What)
42      JBoardman,8.12.2024, created method
43
44      :param first_name: first name
45      :param last_name: last name
46      """
47      self.first_name = first_name
48      self.last_name = last_name
```

**Figure 1.1: Person Class & Constructor**

The Student class includes an attribute of `course_name` and inherits the first and last name attributes from the Person class (Figure 1.2). In the Student class, the constructor `super()` calls the person class to initialize the first and last name data.

```
3 usages
110 class Student(Person): # (jb 8.12.24) student data class
111     """
112     A collection of student data, where person data is inherited
113     ChangeLog: (Who, When, What)
114     JBoardman,8.12.2024, Created Class
115     """
116
117     # (jb 8.12.24) call to the Person constructor and pass it the first_name and last_name data
118     def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''): # (jb 8.12.24) + course_name
119         """
120         Initialize object & inherit from Person class
121         ChangeLog: (Who, When, What)
122         JBoardman,8.12.2024, created method
123
124         :param first_name: first name
125         :param last_name: last name
126         :param course_name: course name
127         """
128         super().__init__(first_name=first_name, last_name=last_name) # (jb 8.12.24) super() brings in the Person data
129
130         # (jb 8.12.24) add an assignment to the course_name property using the course_name parameter (Done)
131         self.course_name = course_name
132
```

**Figure 1.2: Student Class & Constructor**

## Private Attributes & Properties

To ensure attributes are not changed by code outside of the class, an attribute is made private with two underscores preceding the attribute. Then within the class, the attributes are managed by Properties (Figure 1.3). For each attribute, there is a property to get data (accessor) and property to set data (mutator).

```

133     # (jb 8.12.24) added the getter and setter for course_name
134     4 usages (2 dynamic)
135     @property
136     def course_name(self):
137         """
138         Getter for course name with @property decorator
139         ChangeLog: (Who, When, What)
140         JBoardman,8.12.2024, created method
141         :return: formatted name string
142         """
143         return self.__course_name # (jb 8.12.24) private course name
144
145     3 usages (2 dynamic)
146     @course_name.setter
147     def course_name(self, value: str):
148         """
149         Setter for course name
150         ChangeLog: (Who, When, What)
151         JBoardman,8.12.2024, created method
152         :return: course name string
153         """
154         self.__course_name = value # (jb 8.12.24) there is no validation here

```

**Figure 1.3: Properties**

## Body of Script

When implementing data classes and objects, it became necessary to update the processing of dictionary data to and from the JSON into objects. When reading the JSON, the script now loops through the dictionary rows and for each row the Student class is passed the arguments for the first name, last name and course name from which a Student object is created (Figure 1.4).

```

185     try:
186         file = open(file_name, "r") # (jb 8.12.24) read from file
187         # (jb 8.12.24) creating dictionary data from json
188         list_of_dictionary_data = json.load(file) # (jb 8.12.24) the load function returns a list of dictionary rows.
189         for s_record in list_of_dictionary_data: # (jb 8.12.24) Convert the list of dictionary rows into Student objects
190             student_read = Student(first_name=s_record["FirstName"],
191                                   last_name=s_record["LastName"],
192                                   course_name=s_record["CourseName"])
193             student_data.append(student_read)
194         # (jb 8.12.24) close the file
195         file.close()

```

**Figure 1.4: Student class creates Student object from dictionary data**

To prepare for the file output, the opposite processing is performed. The student objects from the student data list are stored as a dictionary so that json.dump can be used to write to the JSON file (Figure 1.5).

```
17         try:
18             list_of_dictionary_data: list = [] # (jb 8.12.24) define the list
19             for s_record in student_data: # (jb 8.12.24) convert List of Student objects to list of dictionary rows.
20                 # (jb 8.12.2024) student_json is a single row class and field (.dot notated) into the dictionary row
21                 student_json: dict \
22                     = {"FirstName": s_record.first_name, "LastName": s_record.last_name, "CourseName": s_record.course_name}
23                 # (jb 8.12.2024) add the row to the dictionary
24                 list_of_dictionary_data.append(student_json)
25             # (jb 8.12.2024) open the file in write mode
26             file = open(file_name, "w")
27             # (jb 8.12.2024) write the dictionary to the file
28             json.dump(list_of_dictionary_data, file)
29             file.close()
30             IO.output_student_and_course_names(student_data=student_data)
```

**Figure 1.5: Student objects to Dictionary**

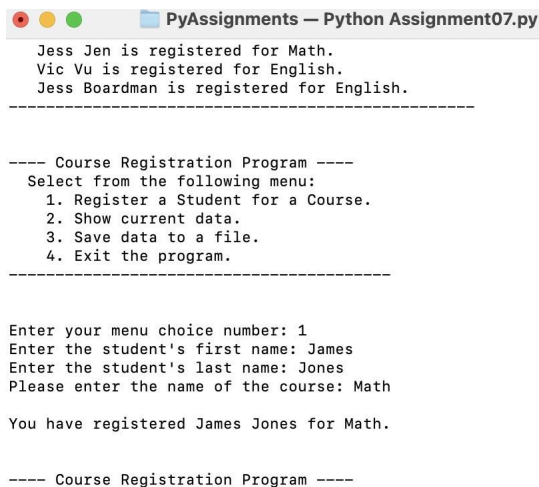
## Testing

The final step in completing Assignment07 was to test the script in both PyCharm and the terminal. To run the program in PyCharm I selected the 'Run' button and was prompted to select from the MENU. Going through each input value, I was able to test the entire script and with the final option '4' I exited the program (Figure 1.6).

```
3. Save data to a file.
4. Exit the program.
-----
Enter your menu choice number: 3
-----
Bob Smith is registered for Python 100.
Sue Jones is registered for Python 100.
Jess Boardman is registered for eng.
Ken Boardman is registered for Math.
James Jones is registered for Science.
Lora Luna is registered for French.
Jess Boardman is registered for gym.
Tommy Tony is registered for French.
James Jones is registered for Gym.
Jess Jen is registered for Math.
Vic Vu is registered for English.
Jess Boardman is registered for English.
-----
---- 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 1.6: PyCharm Testing**

The final testing was completed in the terminal to simulate End User Testing. After opening the terminal, I used the `cd` command to navigate to the correct directory. Once I was in the correct directory, I executed the program by entering `python3 Assignment07.py` in the terminal window. This ran the program and I was able to read the existing file, provide multiple student inputs, display the data and write to the `.json` (Figure 1.7).



```
PyAssignments — Python Assignment07.py
Jess Jen is registered for Math.
Vic Vu is registered for English.
Jess Boardman is registered for English.
-----

---- 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: James
Enter the student's last name: Jones
Please enter the name of the course: Math

You have registered James Jones for Math.

---- Course Registration Program ----
```

### **Figure 1.7: Terminal Output**

## Summary

This week I completed Assignment07 and was able to create data classes which act as templates for working with Objects. The data classes include constructors to initialize the objects and properties to manage the attributes. The processing from and to the JSON file was updated to work between dictionaries and lists of objects. Finally, the script was tested in both PyCharm and the terminal to verify it worked as expected and met all documented acceptance criteria. The script was uploaded to Github for peer review by my classmates.