George Nuesca

November 29, 2023

IT FDN 100 A

Assignment 07

https://github.com/uwgmn/IntroToProg-Python-Mod07

Using Data Classes and Objects

Introduction

This document discusses the implementation of classes in a menu program to further organize data in memory. While a significant amount of reuse was applied from previous assignments, a significant number of additions and changes were made to ensure code fulfills an object-oriented approach. Most of the changes are unbeknown to the user as little is unchanged in how data is formatted in the display and file.

Setting up Data Classes

Data classes for Assignment 07 were set up in several steps. First, the constructor method creates an object, storing attributes for the class in unique addresses in memory. For the *Person* class, these attributes are *first_name* and *last_name*.

Next, Getters and Setters are defined are defined as typical properties for the attributes. Some formatting is applied to the Assignment 07 program using the title() method, which capitalizes the first letter and changes the remaining letters to lower case. There is also some conditional error handling which identifies if characters other than letters are in the name, as well as if the string is empty.

Finally, for the *Person* class, a formatted string is defined to be used to support values if the object is called rather than the address of the object.

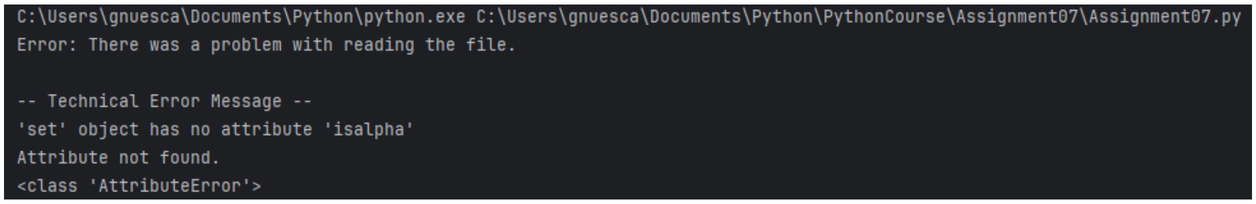
The *Student* subclass is used to define attributes for a person taking a class, inheriting the attributes from the *Person* class. The constructor is extended to accept an additional parameter called *course_name*, with its properties defined within the *Student* subclass. The properties for *first_name* and *last_name* already exist from inheritance of the *Person* class. While not done in this exercise, it is good to note attributes from the superclass, *Person* in this case, can be overridden by the subclass.

Converting Data to Objects (and Vice Versa)

In order to fit the object-oriented approach of the code implementation, data needs to be stored from JSON to attributes defined by the *Student* class. This is done by translating the JSON file to a dictionary, then storing the values into respective *Student* class attributes, which are stored in an object. Note, a dummy variable is used to then append a list of objects corresponding to each student. A loop captures cycles through the translated dictionary data to capture students' information.

Similarly, object attributes can be converted to a dictionary using a loop. The data in the dictionary can then be dumped into a correctly formatted JSON file.

An important aspect of equating attributes to dictionary values is the formatting. As shown in **Figure 1**, if the dictionary value is read, an error is thrown because the attribute can not take any characters other than letters, of which a regular dictionary value includes curly braces. Formatting the dictionary correctly using an f-string alleviates this issue.



```
C:\Users\gnuesca\Documents\Python\python.exe C:\Users\gnuesca\Documents\Python\PythonCourse\Assignment07\Assignment07.py
Error: There was a problem with reading the file.

-- Technical Error Message --
'set' object has no attribute 'isalpha'
Attribute not found.
<class 'AttributeError'>
```

Figure 1: FileProcessor class with collapsed functions

Summary

The primary objective of Assignment 07 was to define data classes to create objects, replacing the dictionary implementation of the menu program. While implementing the defined object classes for input and display arguments was straight forward using dot notation, other aspects of defining objects and the list they reside in required more scrutiny.

For this exercise, debugging using print and type statements helped identify discrepancies in code. Troubleshooting was needed to convert the data between objects and JSON file to adhere to the properties defined in respective attributes.

Separate from the main objective, another minor issue to contend with is naming conventions, which need to be traced accordingly especially in differences between spelling and case sensitivity, such as differences in ”Student” and “students.” Fortunately error messages explaining undefined variable assist in pinpointing the problem.