**Module 8: Objects and Classes**

GitHub Repo:

# Introduction

Week 8 introduces students to objects and classes and their differences. Class patterns and organizations are shown, including the variables (or fields) attributed to it and their visibility. As for objects, students are taught about the constructor and how to give it attributes. Additional topics touched on are methods (both static and private), DocStrings, and how to use GitHub Desktop. The assignment for the week deals with filling out a script made by the instructor using objects and methods that the students must create and fill in.

# Course Content

The author was very familiar with the course content as most of the items discussed are also used by the author when he codes with Java or C#. The week’s content served to distinguish between those languages and how Python does object-oriented programming.

Python classes follow a similar pattern to other object-oriented languages, in that they have fields, methods, and constructors. However, Python has no safeguards to protect its fields or methods, and so Python coders must default to using double underscores to symbolize the visibility of the field (private in this case). Constructors are different in that they keyword self is needed, no visibility modifier is attached, and if the constructor takes a parameter, the type is not specified as it would be with Java. The keyword “this” is also not used in Python. The methods are nearly the same as the constructors, except that there’s also no need to specify if it returns something or not. Java and C# also have static methods though those have to be defined within the name as well.

The DocStrings is similar to the JavaDoc: a short description of the method, its inputs, and outputs if any. Many Java IDEs will auto populate it for constructors and methods, leaving the coder to write the description while already describing the arguments used.

Finally, the author admits to never have used GitHub desktop, preferring the simplicity of uploading directly to the browser. However, he has used Git from this work computer to clone repositories and commit/push code, which allows him to explore all the functions that Git provides.

# Assignment

For this week’s assignment, students were given a “skeleton” script which they had to fill out by using classes, objects, and defined methods. For a great deal of the code, the author referenced the previous assignments as the functionality was essentially unchanged, except for the use of a custom “product” from the defined Product class rather than strings or dictionaries. This section will be split by the different areas where code needed to be implemented.

## Product Class

Students were asked to declare two properties for the class: a product\_name (string) and product\_price (float). The author declared two property methods (aka getters) and a constructor as a global variable indicated that Product-type objects would be stored in it. Figure 1 shows the code for the Product class. The author also took care to preface the properties with the double underscore to indicate that those fields are private to the Product class.

class Product:

"""Stores data about a product:

properties:

product\_name: (string) with the product's name

product\_price: (float) with the product's standard price

methods:

changelog: (When,Who,What)

RRoot,1.1.2030,Created Class

Dennis Negron-Rivera,26/Aug/22,Modified code to complete assignment 8

"""

#pass

# TODO: Add Code to the Product class

#Product Variables

\_\_strPrName = ""

\_\_fltPrPrice = 0.00

#Product Constructor

def \_\_init\_\_(self, name, price):

self.\_\_strPrName = name

self.\_\_fltPrPrice = float(price)

#Getters for access to product variables

@property

def getProductName(self):

return self.\_\_strPrName

@property

def getProductPrice(self):

return float(self.\_\_fltPrPrice)

## Figure 1 Product Class

## FileProcessor Class

Students had to fill out two methods: one to read and load the data from a file, and one to save the data to a file. As in previous assignments, the author used the try-except methods when reading from a file (if the file does not exist, the program crashes so catching the exception allows the program to proceed). When writing to a file, the author kept track of elements with a counter so that newlines were inserted before the last element. File 1 shows the code for the FileProcessor class.



## File 1 FileProcessor Class

## IO Class

The IO class had five items for students to fill out. They must add a doctring, add code to show the menu to user, prompt and get the user’s input, display the items currently on the list, and prompt the user for the product and price. For many of these, the author reused code from previous assignments, including the names and properties (static methods). The author attempted to do input validation and error handling for incorrect inputs. As an example, when getting the user’s input to add a product, if the input validation fails, the system will return a dummy value which the main body will verify. File 2 shows the code from the IO Class.



## File 2: IO Class Code

## Main body of script

For the main body, the instructor had four items that students needed to fill out:

1. Load the data from the file upon program startup.
2. Show the menu of options on the console
3. Get the user’s choice and then implement the actions based on the menu options.

Once again, the author revisited previous assignments and reused code as many of the tasks were similar to those in previous scripts. The only major difference is when adding data, if the dummy value from the IO class is returned, then the script will prevent adding an item to the list; otherwise, it adds it and prints a confirmation message. Additionally, students are not removing items from the list and, when saving the items to a file, they’re also exiting the program. File 3 shows the code for the main body.



## File 3: Main body of script

## Execution

The author ran the script on both VS Code and Terminal for MacOS. Figure 3 shows the run on VS Code; Figure 4, the run on Terminal; and Files 4 and 5, the output for a full run on VS Code and Terminal respectively.

Text

Description automatically generated

## Figure 3: VS Code execution

Text

Description automatically generated

## Figure 4: Terminal Execution



## File 4: VS Code Output



## File 5: Terminal Output

# Summary

This week students learned how to use classes to organize code for object-oriented programming (OOP). The author has extensive experience with classes as Java and C# make use of them, which in turn allow for OOP. The author, however, noted that Python does not mark fields and methods as private, protected, or public and so care must be taken to ensure that they are not accessed incorrectly. The small but noticeable differences between Python and other OOP languages were duly noted.