Lacey Redinger

11/26/19

Foundations of Programming: Python

Assignment 08

GitHubURL

Object-Oriented Programming (OOP)

# Introduction

In Module 8 we learned about creating custom classes and indirectly accessing those classes by instantiating (or creating) object instances of that class. There are various ways you can access and modify an object instance. Generally, this is done as a way to create software objects to be stored as data. Each object from the same class will be essentially the same, but can have different properties. For example, cars can be different colors or styles, but the are all classified as cars, or several squares can be different sizes, but their sides are all equal, classifying them as squares rather than rectangles.

# Understanding OOP

Classes are built with a docstring, a name, some variables and some functions. A class can group ‘like’ functions together, i.e. a class for processing data can have several functions in it which can be called by the *className.functionName.* With OOP, these classes are a little bit special, so variables are called fields, the constructor or initment initiates the creation of the object. Attributes are what define the object, and methods are functions which do things to or with the object. Docstrings help the program author and other programmers understand the intention or purpose of the class.

Accessing a class directly calls a function within that class to preform some task. Accessing a class indirectly creates an object or manipulates an object created by that class. One thing that can be confusing about OOP classes is that they initiate them selves, so they always take “self” as a first parameter. This allows the class to insert itself (attributes, methods, etc.) into the object that is created.

# Properties vs Methods

Functions within a class can have decorators. @property and @staticmethod are 2 examples of decorators. Properties are not really methods anymore because they change the way the object is created, or put restrictions on how the object can be created. The property sort of protects the object and preserves the integrity of the class. @staticmethod allows a “normal” function to exist in a class and cannot be called indirectly.

# Assignment 08

I started by copying a lot of code from my Assignment 07 since my script was really similar to this assignment. Then I also took processing code from Assignment 06. I got all of this to mostly “work” but I was still really confused about how to integrate the main Product Class with the rest of the script. I watched the course video a second time and redid all the labs but I still didn’t really understand the *point.* Why was creating an object from the class different or better than just creating a object from user input? And how could I make the user input actually create the object instances? Chapter 8 in our book kind of helped. I understood how the critters were all “pets” but could be different or have different outcomes depending on how the user interacted with their *Attributes.*

I also watched the Corey Schafer series on OOP (<https://www.youtube.com/playlist?list=PL-osiE80TeTsqhIuOqKhwlXsIBIdSeYtc> **external**) and found them really helpful. I think that my experience was as predicted by Professor Root, that playing around with the code and the classes for several days helped me understand it better. Another thing that helped me this week was actually using and understanding the debugging tools. Additionally, I created my own debugging tool as part of the script and then commented it out when I figured out what was going on.

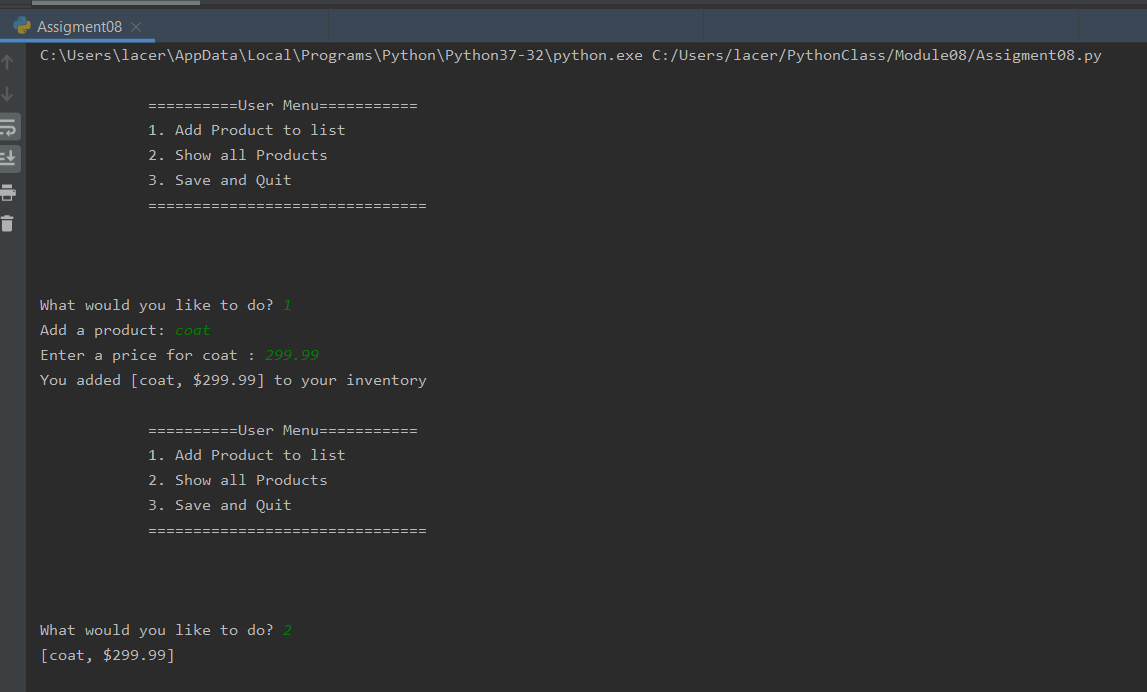
I had a suspicion that some of my issues were sprouting from the fact that I was not actually instantiating objects with my user input, thus not actually using the Product Class at all. To test this I used a counter:

invCount = 0 # used for debugging  
  
def \_\_init\_\_(self, product\_name, product\_price):  
 self.product\_name = product\_name  
 self.product\_price = product\_price  
 Product.invCount += 1 # used for debugging, to make sure objects were instantiated correctly

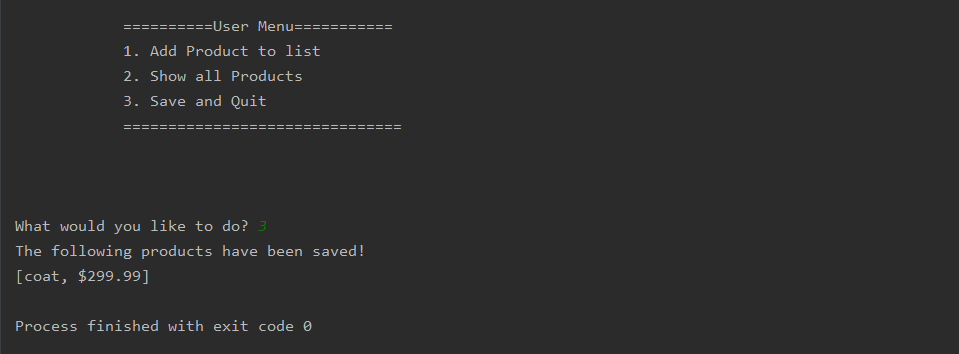
Then I printed Product.invCount after my user input to create the object. When the counter returned that I had zero objects in my list I knew something was wrong. Later on I changed something in my code and it started telling me that I had two objects in my inventory (when I was expecting only one) so I knew that somehow each user input query was creating a separate instance. What was happening is that I was calling Product.product\_name(input(“…?”) and Product.product\_price(input(“…?”)). What I needed to change was create variables to populate the call of the initment. Like so:

product\_name = input("Add a product: ")  
product\_price = input("Enter a price for " + product\_name + " : ")  
  
prodObj = Product(product\_name, product\_price)  
lstOfProductObjects.append(prodObj)

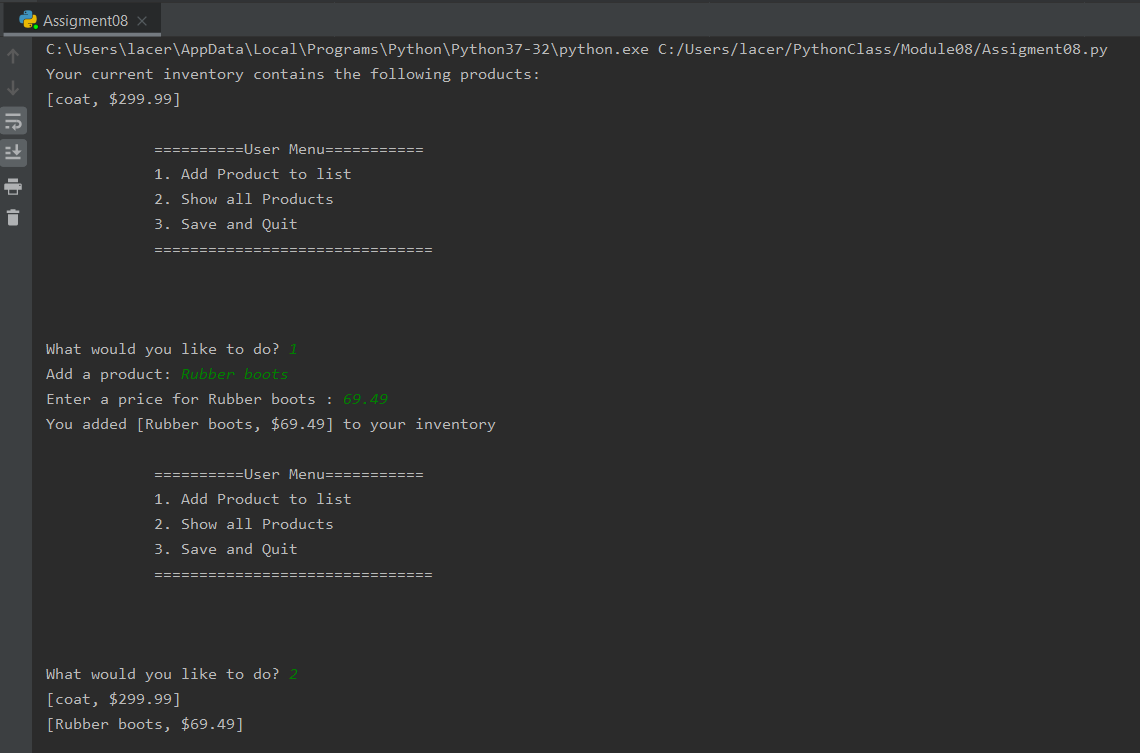
After that, it was a lot of debugging. I had errors with my exception handling, I had errors with saving data to the file, I had errors with formatting. I added an exception loop directly into my get\_user\_choice function which fixed the code crashing, I fiddled around with the other stuff until it worked, and I created a dunder \_\_str\_\_ method in Product Class to format the objects into human readable data.

******

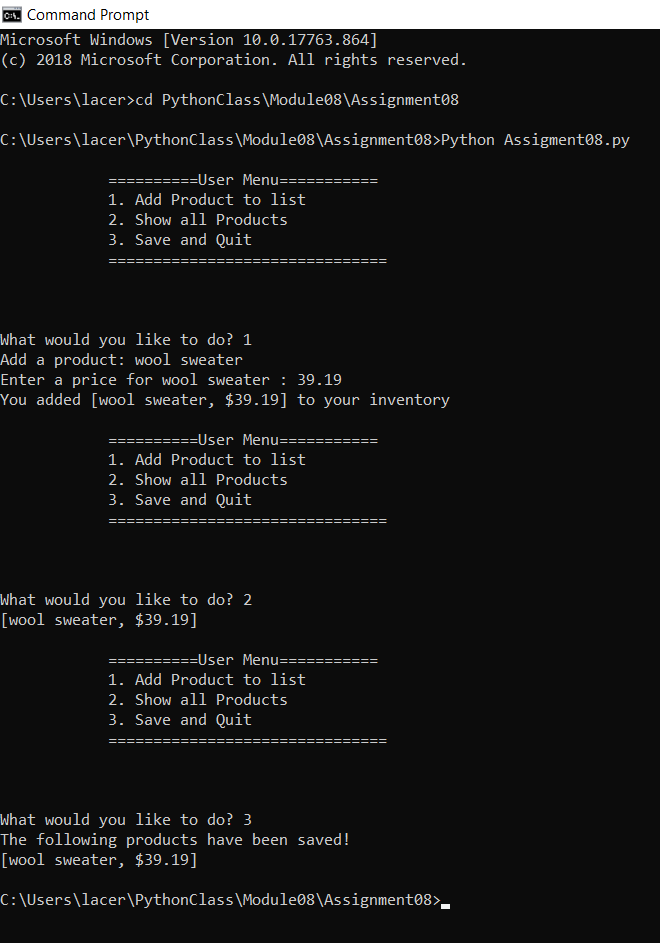
***Figure 1: Script working in PyCharm (Menu Options 1 &2)***

******

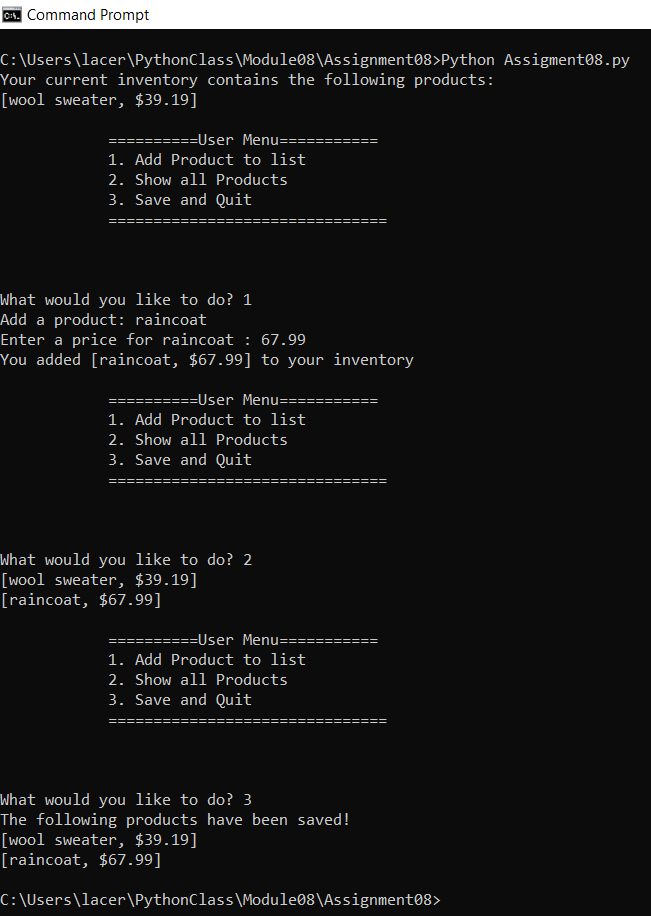
***Figure 2: Script working in PyCharm (Menu Option 3)***



***Figure 3: Script working in PyCharm (Menu Options 1 &2, pulling saved data from file)***



***Figure 4: Script working in Windows CMD: (Menu Options 1-3)***



***Figure 5: Script working in Windows CMD: (Menu Options 1-3 pulling saved data from file)***

# Summary

Creating object instances and putting them into lists is a more complex way to arrange data. I wish I would have had more time to experiment with the capabilities of manipulating objects this way. From the console it looks exactly the same as our earliest modules for creating an Home Inventory list, but with instantiated objects you can create more complex objects with more interesting features.