Angela Asgekar

June 8, 2020

Foundations of Programming: Python

Assignment 08

[GitHub Link](https://github.com/aasgekar/IntroToProg-Python-Mod08)

Object-Oriented Programming: Classes

Introduction

This week was the first introduction to Object-Oriented Programming (OOP) and how to create classes including methods, attributes, properties and constructors. This program had three separate classes that focused on the product as an object, reading/writing to a file and interaction with the user.

Writing the Script

I began by downloading the starter file for the assignment. Since it was broken into three classes and the main body of the program, I decided to work class by class beginning with the Product class. I used a constructor to create two attributes—product\_name and product\_price. I made them private and defined properties so that later on in the program, the user input would have to follow the standards set forth in the properties. Code for the Product class can be seen below.

class Product:  
 *"""Stores data about a product:  
  
 properties:  
 product\_name: (string) with the products's name  
 product\_price: (float) with the products's standard price  
 methods:  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 AAsgekar,6.8.2020,Modified code to complete assignment 8  
 """  
 # --Constructor --* def \_\_init\_\_(self, product\_name, product\_price):  
 *# --Attributes --* self.\_\_product\_name = product\_name  
 self.\_\_product\_price = product\_price  
  
 *# --Properties --* @property  
 def product\_name(self):  
 return str(self.\_\_product\_name).title()  
  
 @product\_name.setter  
 def product\_name(self,value):  
 if str(value).isnumeric() == False:  
 self.\_\_product\_name = value  
 else: raise Exception(**"Product Name cannot be a number."**)  
  
 @property  
 def product\_price(self):  
 return float(self.\_\_product\_price)  
  
 @product\_price.setter  
 def product\_price(self, value):  
 self.\_\_product\_price = float(value)

In the second class, File Processor, I copied functions from my Assignment six and modified the parameters since the functionality was the same. While the starter file only suggested two methods—reading the file and saving to the file—I decided to add a third that would append the new product and its price to a list. Code for the File Processor class can be seen below.

class FileProcessor:  
 *"""Processes data to and from a file and a list of produ  
  
 methods:  
 save\_data\_to\_file(file\_name, list\_of\_product\_objects  
  
 read\_data\_from\_file(file\_name): -> (a list of produc  
  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 AAsgekar,6.8.2020,Modified code to complete assignme  
 """  
  
 # --Methods --* @staticmethod  
 def save\_data\_to\_file(file\_name, list\_of\_product\_objects  
 *""" Saves data from a list of dictionary rows to a f  
  
 :param file\_name: (string) with name of file:  
 :param list\_of\_rows: (list) you want filled with fil  
 :return: feedback statement for the user  
 """* objFile = open(file\_name, **"w"**)  
 for row in list\_of\_product\_objects:  
 objFile.write(row[**"Name"**] + **", "** + str(row[**"Pric** objFile.close()  
 feedback = **"Your data has been saved."** return feedback  
  
 @staticmethod  
 def read\_data\_from\_file(file\_name, list\_of\_product\_objec  
 *""" Reads data from a file into a list of dictionary  
  
 :param file\_name: (string) with name of file:  
 :param list\_of\_product\_objects: (list) you want fill  
 :return: (list) of dictionary rows  
 """* list\_of\_product\_objects.clear() *# clear current dat* objFile = open(file\_name, **"r"**)  
 for line in objFile:  
 name, price = line.split(**","**)  
 row = {**"Name"**: name.strip(), **"Price"**: price.stri  
 list\_of\_product\_objects.append(row)  
 objFile.close()  
 return list\_of\_product\_objects, **'Success'** @staticmethod  
 def add\_data\_to\_list(name, price, list\_of\_product\_object  
 *"""Adds data to a list  
  
 :param name: (string) with name of task:  
 :param price: (string) with name of priority:  
 :param list\_of\_product\_objects: (list) you want to a  
 :return: (list) of dictionary rows  
 """* dicRow = {**"Name"**: name, **"Price"**: price}  
 list\_of\_product\_objects.append(dicRow)  
 return list\_of\_product\_objects, **'Success'**

The IO class was pretty simple since the methods were either gathering user input or displaying data to the user. I chose to break out the inputs for the product name and price into two methods as opposed to one so that later in the program it would be easier to check for exceptions if the user input didn’t match the defined properties. Code for the IO class can be seen below.

class IO:  
 *"""Displays output to user and receives user inputs:  
  
 methods:  
 menu():  
  
 menu\_choice(): -> (string)  
  
 print\_current\_inventory(list\_of\_product\_objects):  
  
 user\_input\_product\_name(): -> (string) name  
  
 user\_input\_product\_price(): -> (string) price  
  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 AAsgekar,6.8.2020,Modified code to complete assign  
 """  
 # --Methods --* @staticmethod  
 def menu():  
 *""" Displays menu to the user  
  
 :return: nothing  
 """* print \  
 (**"""  
   
 Product Inventory  
   
 1 - View current product inventory  
 2 - Add a product to inventory  
 3 - Save data and exit program  
   
 """**)  
  
 @staticmethod  
 def menu\_choice():  
 *""" Gets user's menu selection  
  
 :return: (string) choice  
 """* choice = input(**"Please select a menu option: "**)  
 return choice  
  
 @staticmethod  
 def print\_current\_inventory(list\_of\_product\_objects):  
 *""" Shows the current Tasks in the list of diction  
  
 :param list\_of\_product\_objects: (list) of rows you  
 :return: nothing  
 """* if list\_of\_product\_objects == []:  
 print(**"There is currently nothing in the Produ** else:  
 print(**"The current Product Inventory is:"**)  
 for row in list\_of\_product\_objects:  
 print(row[**"Name"**] + **" ("** + str(row[**"Price"** @staticmethod  
 def user\_input\_product\_name():  
 *""" Gets user input for Product Name  
  
 :return: (string) name  
 """* name = input(**"Enter Product Name: "**)  
 return name  
  
 @staticmethod  
 def user\_input\_product\_price():  
 *""" Gets user input for Product Price  
  
 :return: (string) price  
 """* price = input(**"Enter Product Price: "**)  
 return price

Lastly, the main body of the code is in infinite loop that breaks if the user chooses menu item three which saves the data to the file and exits the program. I have two try and except statements to check for exceptions. The first tries to load the current data in the file into the list and if an error is received because there is no file, the file is created. The second, I used to see if any errors would be generated from the user inputs not matching the property definitions. It first checks to make sure the name is not a number and then if it is not, it checks to make sure the price is a number. If both of the conditions were met, it appends the information to the list. Code for the main body of the program can be seen below.

try:  
 FileProcessor.read\_data\_from\_file(strFileName, ls  
except FileNotFoundError:  
 *#create the file if it didn't already exist* FileProcessor.save\_data\_to\_file(strFileName, lstO  
  
  
user\_choice = **" "**while user\_choice != None:  
*# Show user a menu of options* IO.menu()  
*# Get user's menu option choice* user\_choice = IO.menu\_choice()  
  
*# Show user current data in the list of product objec* if user\_choice == **"1"**:  
 print()  
 IO.print\_current\_inventory(lstOfProductObject  
 input(**"Press enter to return to the menu."**)  
  
*# Let user add data to the list of product objects* elif user\_choice == **"2"**:  
 print()  
 p1 = Product(**"Product"**, 0)  
 try:  
 *#test that input obeys properties* p1.product\_name = IO.user\_input\_product\_n  
 except Exception as e:  
 print(e)  
 input(**"Press enter to return to the menu.** continue  
 try:  
 *#test that input obeys properties* p1.product\_price = IO.user\_input\_product\_  
 FileProcessor.add\_data\_to\_list(p1.product  
 print(**"Your data has been added to the li** input(**"Press enter to return to the menu.** except ValueError:  
 print(**"Price entered was not a number."**)  
 input(**"Press enter to return to the menu.** continue  
  
*# let user save current data to file and exit program* elif user\_choice == **"3"**:  
 print()  
 print(FileProcessor.save\_data\_to\_file(strFile  
 print(**"Thank you for using the program."**)  
 input(**"Press enter to exit."**)  
 break  
  
 else:  
 print()  
 print(**"That was not a menu item."**)  
 print(**"Please select 1, 2 or 3."**)  
 input(**"Press enter to return to the menu."**)

Testing the Script

I tested various methods throughout writing the function in PyCharm. At the end, once I knew individual components work, I tested it all together to check for exceptions or any display errors I wanted to address. An example of the program being tested in PyCharm can be seen in Figure 1.

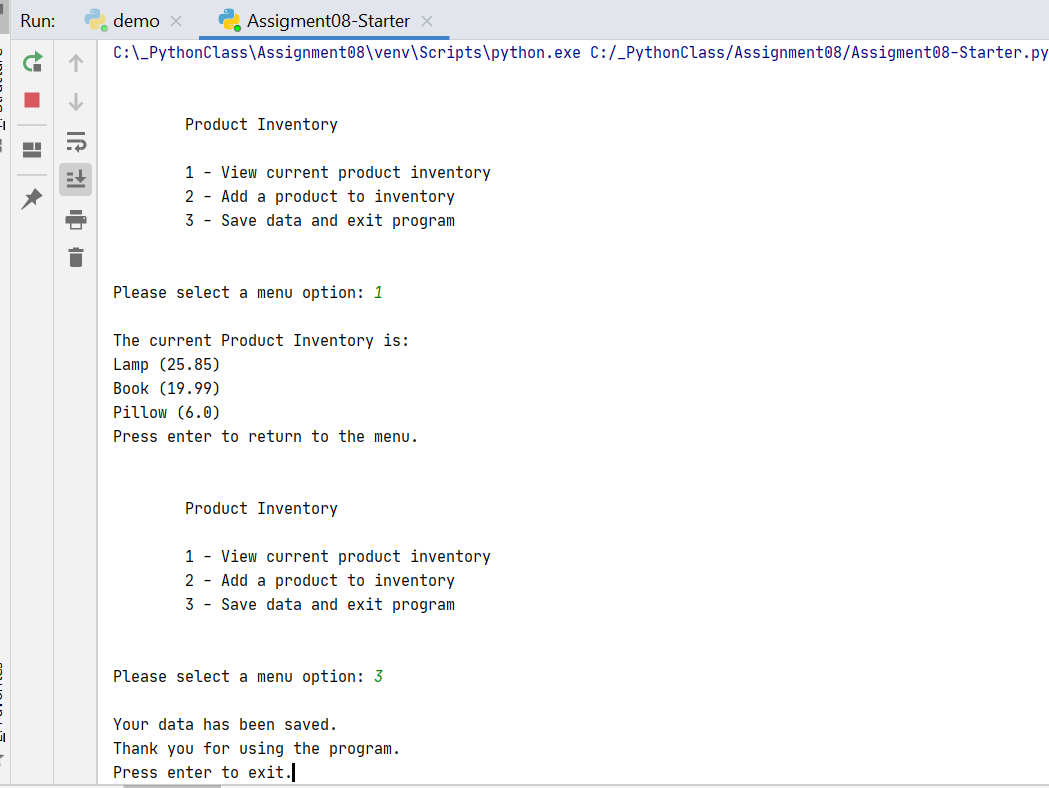


Figure 1: Testing in PyCharm

I then tested the program in the Console (Figure 2)

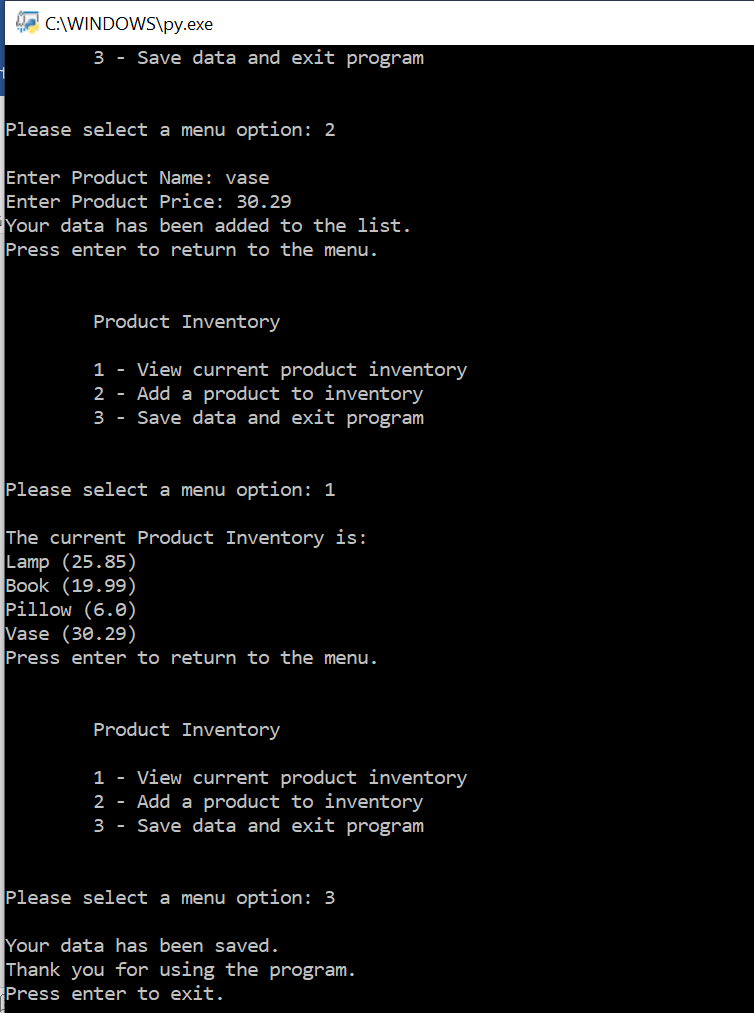


Figure 2: Testing in the Console

Summary

The concept of Object-Oriented Programming was challenging to work through and fully understand. Throughout the assignment I felt there were likely better ways I could perform the actions, but didn’t grasp an understanding of the concept well enough to do so. I was able to use attributes, properties and methods to perform the program task and gain a better understanding of classes.