

Class and Objects

Content Outline

| | |
|------------------------|-----|
| Introduction | 1 |
| Class vs Objects | 1 |
| Class Components | 2-3 |
| Fields | |
| Constructor | |
| Getter/Setter | |
| Methods | |
| Script | 3-6 |
| GitHub Desktop..... | 6-7 |
| Summary | 7 |
| Resources | 7 |

Introduction

The assignment for this week was to create a program that took in user inputs for product name and product price, and then give the user the option to display current items, add more items, or save items to file. Here we will demonstrate how to do that, with the use of class-objects, getter/setter properties, constructors, to name some.

Class vs Objects

How is a class different from an object? A class is a way to group together variables and functions under a name, by which it can be called out to in later code. A simple way to describe an object is to think of it as a “copy” of a class; it is a reference variable to the class. Earlier in this course we learned that a class can just have variables and methods, we’ll dig a little deeper into some other components that (can) make up a class.

Class Components

Fields

Fields are variables that are defined under a class; direct nesting under the class and NOT under any methods. However, not all classes have to have fields, as sometimes the fields can remain hidden or privatized.

Constructor

A constructor is a specialized method that is often used to set the initial values of Field data. This method is also known as the “initialization method.” It is easily recognized by `__init__`. An example can be seen in the script (Figure 1).

```
def __init__(self, product_name: str, product_price: float):  
    """ Getting name of product and price """  
    #--Attributes--#  
    try:  
        self.__product_name = str(product_name) #privatize attribute using __  
        self.__product_price = float(product_price)  
    except Exception as e:  
        raise Exception("Problem with types of values entered: ")  
        print(str(e))
```

Figure 1. Constructor or initialization method with attributes nested under a try-except block. Attributes are `product_name` and `product_price`.

This constructor is nested under the class “Product”. The constructor allows for the class to be referenced by using “self” to grab the necessary attributes to for minimum functionality.

Properties

Properties are also specific methods. Two properties under a class are the getter and setter. A getter allows for the reference/indirect access of class attributes without having the user messing it up; it allows for code to be preserved.

Another property is the setter, which sets the value of the getter. For these properties, it is important to have the getter precede the setter.

```
@property #getter  
def product_price(self):  
    return float(self.__product_price)  
  
@product_price.setter #setter  
def product_price(self, value: float):  
    if str(value).isnumeric():  
        self.__product_price = float(value)  
    else:  
        raise Exception("Invalid. Price should not have letters")
```

Figure 2. Examples of getter and setter properties within the program code.

Methods

We have worked with these before! Methods can be identified as any function defined under a class. Variables that are nestled under a method are called *attributes*. The try-except block in Figure 1 is considered an attribute.

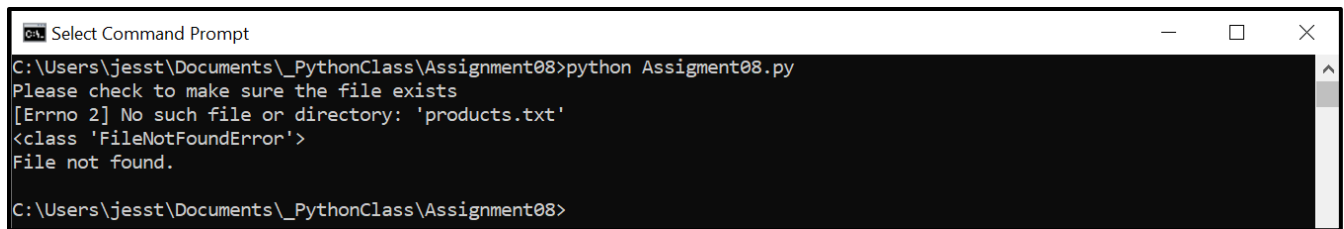
The Script

The script for this assignment is quite long (so below I've just included the main body). In my script, I added code to three classes—Product, FileProcessor, and IO—from which I am able to reference class methods in the main body. Essentially, the Product class stores user product-related input, the FileProcessor class stores and reads file to and from the text file, respectively, and the IO class encapsulates all functions related to data display and direct user input. As the user goes through the program, they are able to select a choice (1-4) and perform a menu function (Figure 3).

```
191 # Main Body of Script ----- #
192
193 # Load data from file into a list of product objects when script starts
194 try:
195     lstOfProductObjects = FileProcessor.read_data_from_file(strFileName)
196
197     while True:
198         # Show user a menu of options
199         IO.MenuOptions()
200         # Get user's menu option choice
201         strChoice = IO.UserChoice()
202         # Show user current data in the list of product objects
203         if strChoice.strip() == '1':
204             IO.CurrentData(lstOfProductObjects)
205             continue
206         # Let user add data to the list of product objects
207         elif strChoice == '2':
208             lstOfProductObjects.append(IO.GetProduct())
209             continue
210         # Let user save current data to file and exit program
211         elif strChoice == '3':
212             FileProcessor.save_data_to_file(strFileName, lstOfProductObjects)
213             continue
214         # Let user exit program
215         elif strChoice == '4':
216             input("Thanks for using this program. Goodbye. \n(Press 'Enter' to exit)")
217             break
218         # if input not 1-4, ask user to enter choice again
219         else:
220             ("Please enter a valid number form the menu")
221
222     except FileNotFoundError as e:
223         print("Please check to make sure the file exists")
224         print(e, type(e), e.__doc__, sep='\n')
225
226     except TypeError as e:
227         print(e, e.__doc__, type(e), sep='\n')
228
229     except AttributeError as e:
230         print("The object or class cannot perform attribute")
231         print(e, type(e), e.__doc__, sep='\n')
232
233     except Exception as e:
234         print(e, type(e), e.__doc__, sep='\n')
235
```

Figure 3. (Above) Main body—while loop nestled within try-except block—with references to class methods (not shown).

To verify that the script works, I tested it in the command line (Figure 4).

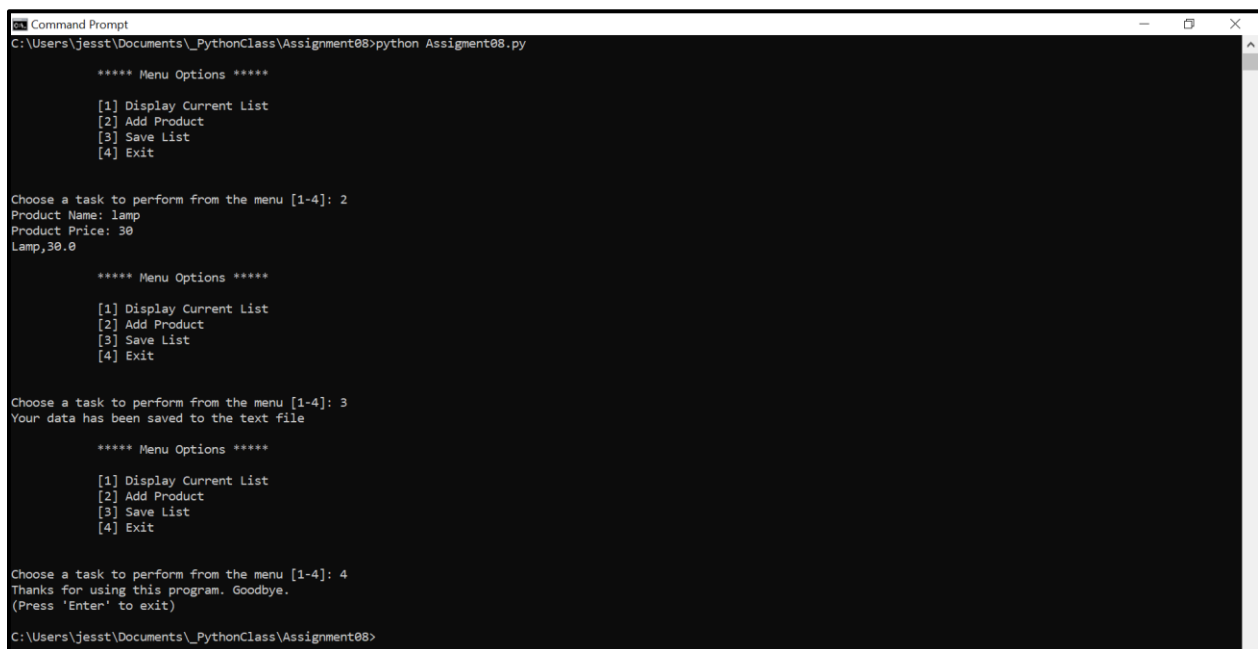


```
C:\Users\jesst\Documents\_PythonClass\Assignment08>python Assignment08.py
Please check to make sure the file exists
[Errno 2] No such file or directory: 'products.txt'
<class 'FileNotFoundError'>
File not found.
C:\Users\jesst\Documents\_PythonClass\Assignment08>
```

Figure 4. Program executed in command line. Error message produces from first except block in main body of Figure 3.

As you can see, the binary file did not exist, so a message was produced from the first try-except block in the main body (Figure 3). Once corrected (by adding a blank text file by the name “products.txt”), the program is able to find the file and continue to ask the user for input(s).

Here we see the command prompt output after the text file was created. The program runs smoothly.



```
Command Prompt
C:\Users\jesst\Documents\_PythonClass\Assignment08>python Assignment08.py

***** Menu Options *****

[1] Display Current List
[2] Add Product
[3] Save List
[4] Exit

Choose a task to perform from the menu [1-4]: 2
Product Name: lamp
Product Price: 30
Lamp,30.0

***** Menu Options *****

[1] Display Current List
[2] Add Product
[3] Save List
[4] Exit

Choose a task to perform from the menu [1-4]: 3
Your data has been saved to the text file

***** Menu Options *****

[1] Display Current List
[2] Add Product
[3] Save List
[4] Exit

Choose a task to perform from the menu [1-4]: 4
Thanks for using this program. Goodbye.
(Press 'Enter' to exit)
C:\Users\jesst\Documents\_PythonClass\Assignment08>
```

Figure 5. Program output in command line after products.txt file created.

Furthermore, the program is ran in the Pycharm IDE console (Figure 6), and is seen to have ran smoothly.

```
C:\Users\jesst\anaconda3\envs\Assignment08\python.exe C:/Users/jesst/Documents/_PythonClass/Assignment08/Assignment08.py

**** Menu Options ****

[1] Display Current List
[2] Add Product
[3] Save List
[4] Exit

Choose a task to perform from the menu [1-4]: 1

***** Your Current List: *****
Lamp,30.0
*****

**** Menu Options ****

[1] Display Current List
[2] Add Product
[3] Save List
[4] Exit

Choose a task to perform from the menu [1-4]: 2
Product Name: notepad
Product Price: 4
Notepad,4.0

**** Menu Options ****

[1] Display Current List
[2] Add Product
[3] Save List
[4] Exit

Choose a task to perform from the menu [1-4]: 3
Your data has been saved to the text file

**** Menu Options ****

[1] Display Current List
[2] Add Product
[3] Save List
[4] Exit

Choose a task to perform from the menu [1-4]: 4
Thanks for using this program. Goodbye.
(Press 'Enter' to exit)

Process finished with exit code 0
```

Figure 6. Script executed in the IDE PyCharm console.

To confirm that the end-user's data is saved to the text file in the appropriate directory, we can go to the file directory and open the text (Figure 7). We can see that upon opening the .txt file in a notepad, that the data saved is from the program being executed in the command line (Figure 5) and in the PyCharm console (Figure 6). The "lamp" data is saved from the command line and the "notepad" data is saved from the PyCharm console.

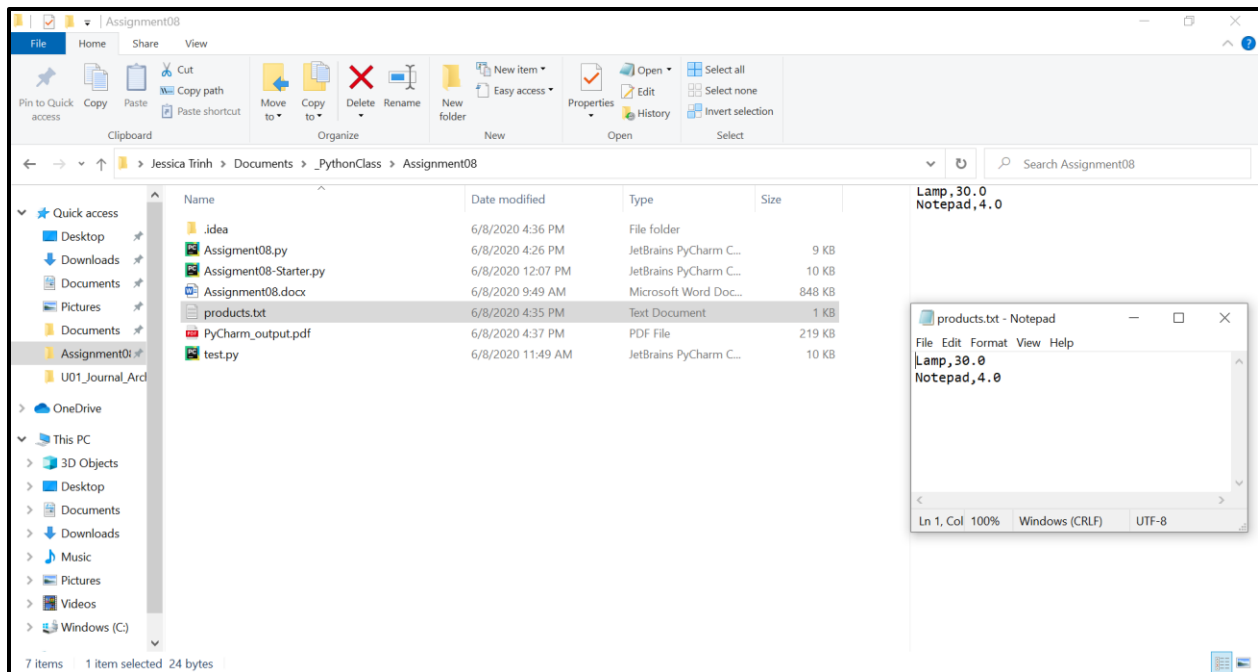


Figure 7. Text file products.txt located in program directory. File opened with data from both program executes saved.

GitHub Desktop

Unfortunately, I forgot to capture images along the way, but one way to create a new repository on GitHub is to use the GitHub desktop application, where you can select to create a new repository, copy files and commit those files. Once committed, you have to push the files.

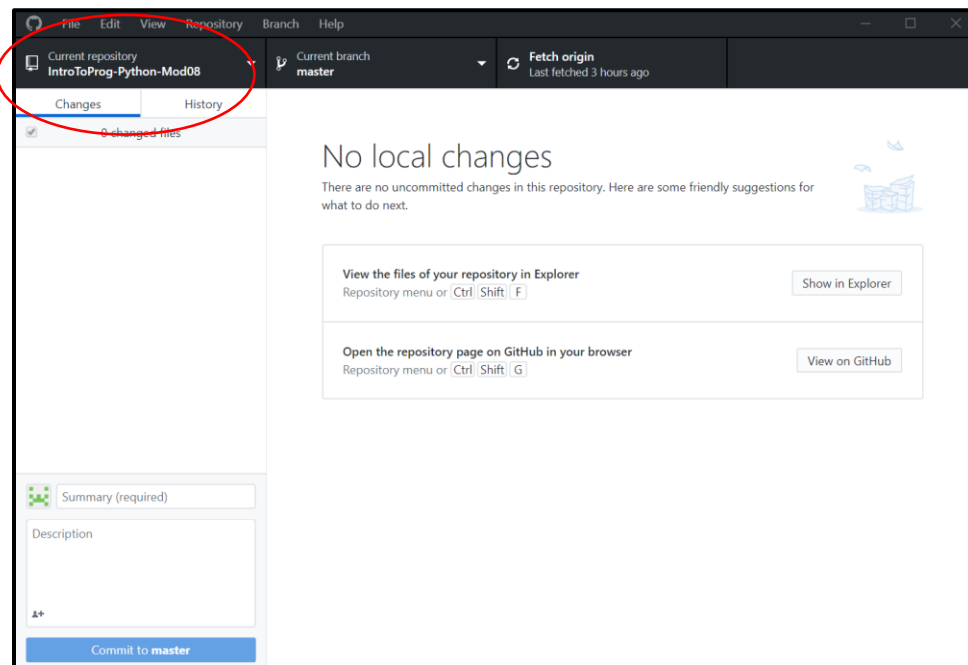


Figure 8. Aftermath of pushing files using GitHub desktop application.

As you can see, a repository was successfully created (circled in red) using the desktop application (Figure 8). Checking my repository online, I can see that the repository creation has been synced (Figure 9).

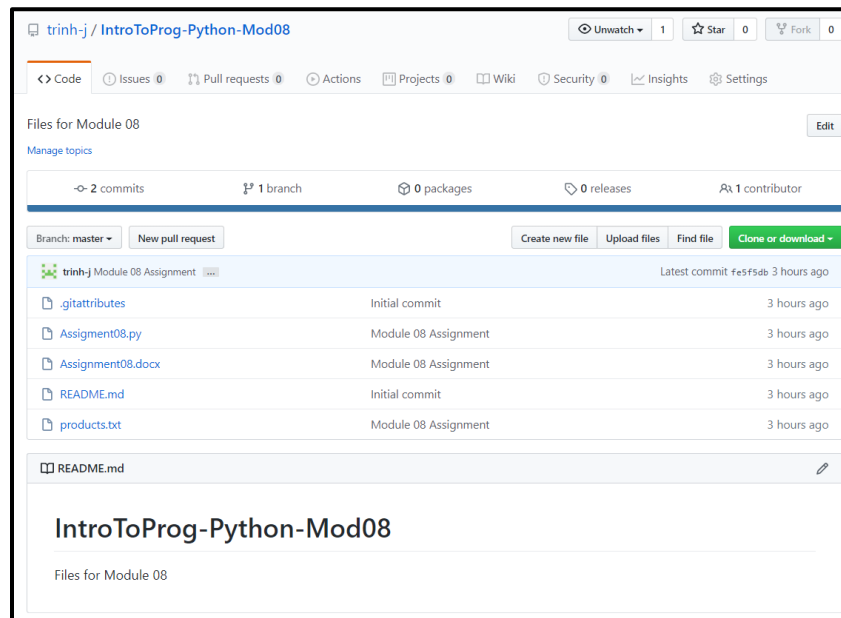


Figure 9. Aftermath of pushing files using GitHub desktop application-repository visible online

At first when I committed and pushed the files using the desktop app, I accidentally kept my repository private. I was able to change this online when I went to settings and changed settings from private to public.

Summary/Discussion

For this assignment, we had to implement specialized class-methods into our code which allowed us to properly reference fields and attributes while preserving the functions. The assignment could have been done without the specialized methods. I found that this assignment helped clear up my understanding of constructors, getters/setters, and privatizing classes/attributes; though I still have some trouble understanding when to privatize certain objects. There was a lot of good information from our course resources that I referenced (Module 8 notes and the textbook and lecture videos), and I couldn't seem to get the best answer for understanding these topics to the fullest. Though it seemed that this should have been intuitive, I found myself questioning whether or not I'm using this correctly, and thus took a very long time completing this assignment. This was a tough assignment. Nonetheless, I hope you enjoyed this work.

Resources

1. Module08 Programming Notes (Root)
2. (Chapter 7 in the course textbook)
3. <https://www.datacamp.com/community/tutorials/property-getters-setters>