Molly Weaver

March 06, 2021

Foundations of Programming: Python

Assignment 08 – Object Oriented Programming

Object-Oriented Programming

# Introduction

For homework 08 we built on the To Do List program from HW06. We added in the pickling and error handling concepts learned in HW07 and added a new concept – object-oriented programming.

## The Program

We were again given a starter program from the professor, mostly consisting of pseudo-code, and were asked to fill in the rest of the code.

I started with collecting input from and displaying output to the user. I re-used a lot of my code from HW06 with adaptations. First, I created a menu of options to be displayed to the user (**figure 1**).

Graphical user interface, text, application

Description automatically generated

***Figure 1.*** Displaying menu options to the user.

I did change some of the options later on – specifically I removed the option to delete an item. I ran into some issues, then ran out of time, and decided to reduce the scope of my program.

Then I called the function from the main program (**figure 2**).

Table

Description automatically generated with low confidence

***Figure 2.*** Calling the menu function.

I admit I tried to run the program at this point. Of course, I had created an endless loop as I had no stopping points yet. That was a learning experience.

I copied the method to capture the user’s menu option from HW06 and added a call statement in the main body. I also declared the variable used to capture the data in my data section. Before testing the program again, I added an if/else block and filled out code for option five to exit the program, hoping to avoid another endless loop (**figure 3**).

Text

Description automatically generated

***Figure 3.*** Adding code to avoid the endless loop.

Running the program again gave me the expected output (**figure 4**).

Text

Description automatically generated

***Figure 4.*** Capturing the user’s choice.

Next, I started developing my product class using the lecture 08 labs. I went through them one by one building each time to be sure everything was working properly, since I will admit, my understanding is still a little shaky (**figure 5**).

Graphical user interface, text, application, email

Description automatically generated

***Figure 5.*** Building the Product class, step 1.

And checking after each change to be sure my output was as expected.

Text

Description automatically generated with medium confidence

***Figure 6.*** The expected output.

**Figures 7-9** show the continuation of this step-by-step process.

Graphical user interface, text

Description automatically generated

***Figure 7.*** Building the Product class, step 2.

Text

Description automatically generated

***Figure 8.*** Building the Product class, step 3.

Text

Description automatically generated

***Figure 9.*** Building the Product class, step 4.

I decided to wait and add more detail on error handling later after the rest of the program was working.

I then transferred everything from my test environment into my main program and re-ran the program to check that it was still working (**figure 10**).

Text

Description automatically generated

***Figure 10.*** Still getting the same output.

Next, I added the code to add the new item to a list (**figures 11-13**).

Graphical user interface, text

Description automatically generated with medium confidence

***Figure 11.*** Adding to the list – main function call.

Text

Description automatically generated

***Figure 12.*** Adding to the list – the function.

Text

Description automatically generated

***Figure 13.*** Adding to the list – the output.

All this code was taken from HW06 with a few modifications.

The next step was to save the data to a file. Instead of using the method from HW06, I used my pickling program from HW07, but put it into a function (**figures 14-16**).



***Figure 14.*** Pickling the data to a file – main function call.

Text

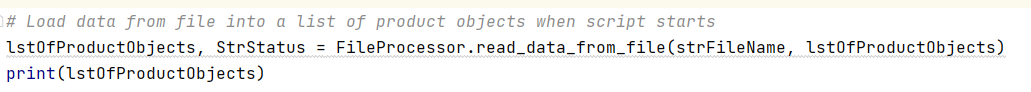
Description automatically generated

***Figure 15.*** Pickling the data to a file – the function.



***Figure 16.*** Pickling the data to a file – the output.

The data was saved to the file in binary, as expected. The next step was to read the data back out of the file (**figures 17-19**).



***Figure 17.*** Unpickling the data from a file – main function call.

Text

Description automatically generated

***Figure 18.*** Unpickling the data from a file – the function.



***Figure 19.*** Unpickling the data from a file – the output.

The final menu option to write was to print the current product list. I again took my code from HW06 (**figures 20-22**).



***Figure 20.*** Printing the current product list – main function call.

Text

Description automatically generated

***Figure 21.*** Printing the current product list – the function.

Text

Description automatically generated

***Figure 22.*** Printing the current product list – the output.

I then went back to my Product class to try and add in some input formatting and error handling. My basic understanding of OOP is, first you create an instance of an object in the main program (**figure 23**).

Text

Description automatically generated

***Figure 23.*** The revised code to create an instance.

I moved my code around a bit so that the user input was gathered first, then the instance was created. This caused the \_\_init\_\_ method to be executed (**figure 24**).

Text

Description automatically generated

***Figure 24.*** The \_\_init\_\_ method.

Then I added the attributes strProduct and strPrice back in the main program (**figure 23**). I changed the Price variable to a string to avoid confusion for the time being. Back in the Product class, the getter method was executed (**figure 24**), in this case, for the price attribute.

Text, letter

Description automatically generated

***Figure 24.*** The getter method.

This was where any formatting of the attribute could be done if required. Then, the setter method was executed (**figure 25**).

Graphical user interface, text

Description automatically generated

***Figure 25.*** The setter method.

This was where I tired to add in error handling. I wanted to ensure that the price only contained numbers (**figure 26**).

Text

Description automatically generated

***Figure 26.*** Rejecting “faulty” user input.

It looked like it was functioning as I expected, but I got the same error message with a numeric input (**figure 27**).

Text

Description automatically generated

***Figure 27.*** Rejecting “good” user input.

I struggled with this for a long time and couldn’t come up with something that worked properly. Then I realized that I also had no idea how to kick the user back to the input function if they did enter what I considered faulty data. I decided to put the error handling into a static function until I learned more about OOP.

I wrote a new static method to check that the user input for price only contained numeric characters (**figure 28**).

Graphical user interface, application

Description automatically generated

***Figure 28.*** Error handling in a static method.

If the price didn’t contain only numbers, the method returned “wrong” and if it contained anything else it returned “right.” I confess this wasn’t the most exhaustive of error handling. Then in the main program I used the return value of “right” to add the new object to the list (**figure 29**).

Text

Description automatically generated

***Figure 29.*** Using the return value from the error handling method.

If it returned “wrong,” the menu would re-appear. The two different outcomes are shown in **figures 30-31** below.

Text, letter

Description automatically generated

***Figure 30.*** Rejecting “faulty” user input.

Scatter chart

Description automatically generated with low confidence

***Figure 31.*** “Good” input is added to the list.

As a last step, I ran the program using the command line and everything seemed to function properly.

### Conclusion

This week we built on the HW06 assignment but used the program to create a list of products and prices instead of a To Do list. A lot of the code was similar. I took advantage of that and copied in a lot of the functions from HW06. We also added in the concepts learned from HW07 – pickling and error handling. So instead of needing to unpack the list of dictionaries to write the data into a text file, I could just pickle it into a binary file. I wrote a new method to ensure the user entered a number for the price using error handling. I confess that my exceptions could have been more detailed, but I was happy that it worked in the basic way it was set up. The final twist this week was to use object-oriented programming. I feel like I got a very basic, surface level understanding from class, but not really enough to do more than mimic what the professor showed us. I’m not entirely sure I used it in the way it was intended, but it functioned. Hopefully we will do a deeper dive in the coming weeks.