Justin Calhoun

Professor Summer Rae Elasady

Foundations Of Programming: Python

11 November 2018

Assignment 06 - ToDo List, Functions and Class

This week’s assignment was to take the code we wrote for Assignment 05 and refactor it so that the majority of the logic would be encapsulated into a series of functions within a class. Therefore, I will focus less on what the code does this time, and instead on the changes required to encapsulate it.

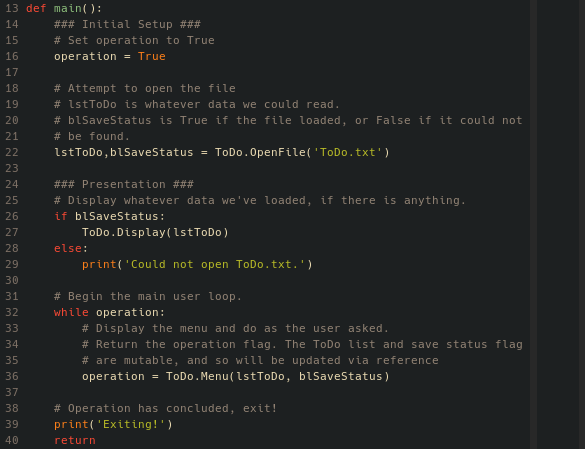


Figure 1: main() function

The new main() function is particularly baren now, with nearly everything moved into the class methods. At the start, I set a new boolean variable, operation, to True. This is used as the condition in the while loop, and once the user chooses to exit, it will be set to False and end the loop. Next, I call my ToDo.OpenFile() method, passing in the file name “ToDo.txt” and returning the list and a flag for the save status. Then, if I was able to read something from the file, I call ToDo.Display() to display the contents of the list.

Finally, the while loop begins, and consists of a single line. I call ToDo.Menu(), which does the real work of the script. I pass in the ToDo list and the save status flag, and return the operation state. From this point on in execution, lstToDo and blSaveStatus will continue to be used as arguments in the various other methods, and due to their mutable nature, the original variables in memory will be updated without my explicitly returning them.

All of the remaining functions in this script are methods of a class named ToDo, and I’ll now go through them in the order they are written in the script itself: which is, to say, no particular order.



Figure 2: ToDo.OpenFile() method

The ToDo.OpenFile() method is fairly simple, as it takes in a filename as a string, and then attempts to open that file and read it into a list of dictionary key-value pairs. In order to encapsulate this from last week’s version, I removed hard references to “ToDo.txt” so that it could theoretically be used on any file in the future, and added return statements to pass back out the ToDo list and the save status flag.



Figure 3: ToDo.SaveFile() method

The ToDo.SaveFile() method was already a function in my code last week, but was far simpler. In Assignment 05, it was focused entirely on the act of saving the file, and the user input asking to confirm the save was in the main() function prior to SaveFile() being called. This week, it gained that user interaction code, as well as a new parameter, savestatus, so that the flag can be updated during the save operation. It still takes in the filename and the ToDo list, and uses these parameters identically to last week.



Figure 4: ToDo.Display() method

The ToDo.Display() method was also already a function last week, and in this case, received no changes at all.



Figure 5: ToDo.Menu() method, part one

The ToDo.Menu() method ended up with the bulk of what was in the main() function last week. It takes the ToDo list and save status flag as parameters, and then passes them through to whichever methods need them when the user calls them. It begins by defining the menu items as a list of dictionaries, and then iterates over it to display the options and ask the user to select one.



Figure 6: ToDo.Menu() method, part two

Then, depending on how the user answered, it will call the other methods to perform the various actions for the user. At the end of it’s execution, if the user chose options 1-4, the method will return True to main()’s operation variable, keeping the loop going. If the user chose 5, it will return False to end the while loop.



Figure 7: ToDo.AddDel() method, part 1

The ToDo.AddDel() method should probably be two methods, but I chose to read the homework requirements literally. When calling this method, it accepts the ToDo list, save status flag, and an operation as parameters. If it is called with “add”, it will run the same code as last week to ask the user for input and append the new dictionary to the ToDo list.



Figure 8: ToDo.AddDel() method, part two

If called with “del”, it runs the same code from last week to confirm and remove an item from the ToDo list. At the end, it returns nothing, as all work done will occur directly on the highest-scope variables thanks, again, to mutable variables as arguments being references to the same location in memory.

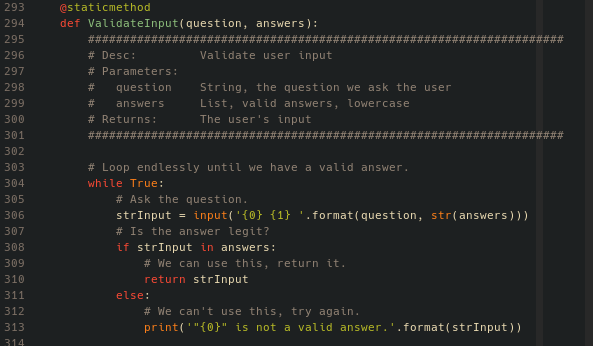


Figure 8: ToDo.ValidateInput() method

The final method is ToDo.ValidateInput(), which was also a full function last week, and was not changed at all.

The biggest challenge this week was wrapping my head around arguments as references to the original variable. Prior to this class, I had no idea this was the behavior in python, and had assumed when writing functions that I needed to explicitly pass everything I wanted in and out. However, in the first version of this week’s script, I “forgot” to pass the ToDo list back out at one point, and when I realized this, was confused as to how the script was still functioning. After additional research and re-reading the book and linked pages, I finally “got it”, and decided to go back and refactor the code to lean hard into this behavior, only explicitly returning things I needed to, and letting the reference behavior take care of the rest.

Philosophically, though, I’m not sure how I feel about it. Explicitly passing variables back out feels “cleaner”. I’ll be doing some more research into how other people leverage or ignore this behavior in python before deciding how I want to utilize it going forward.