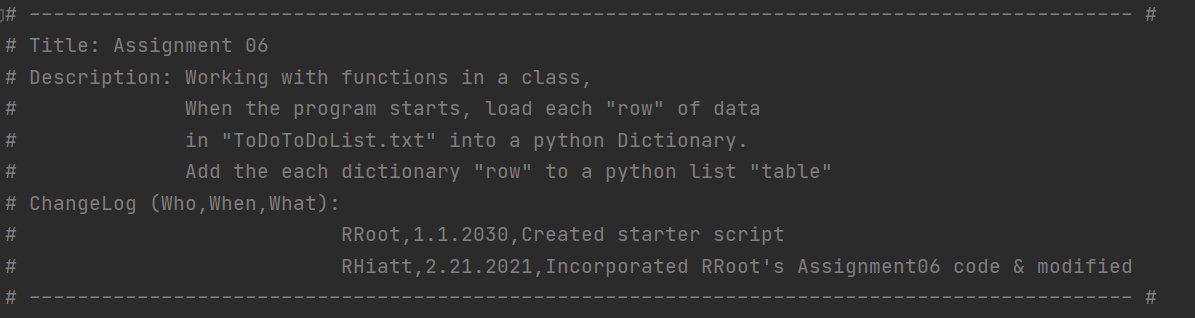
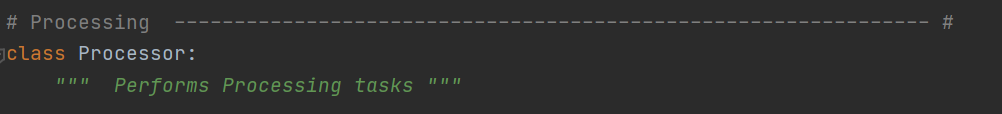
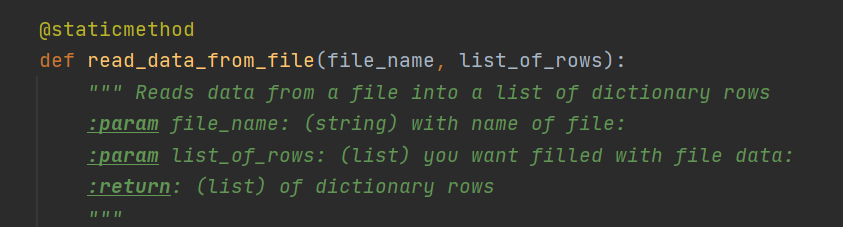
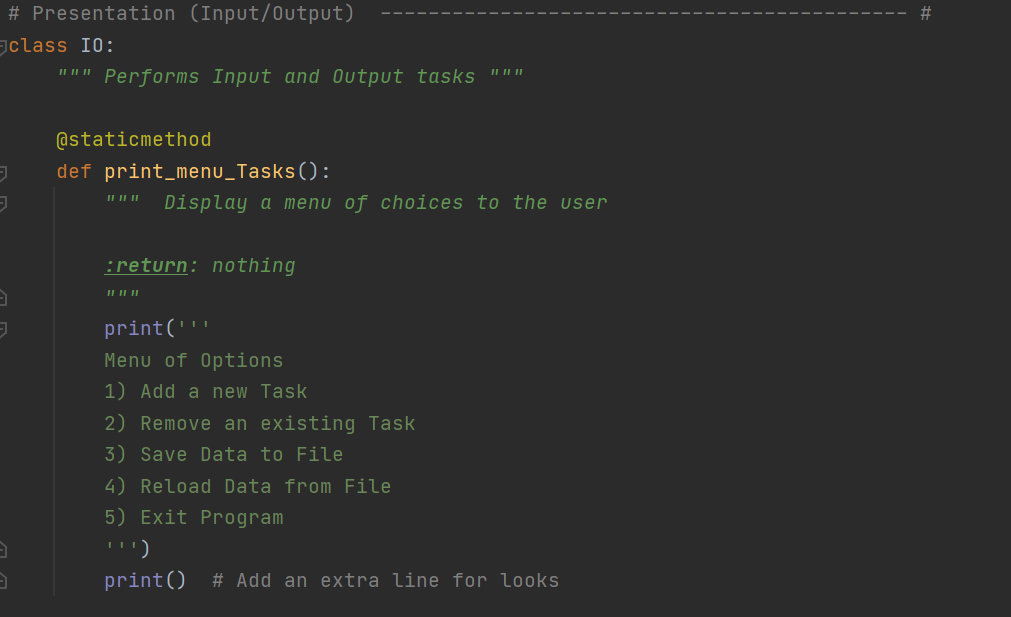
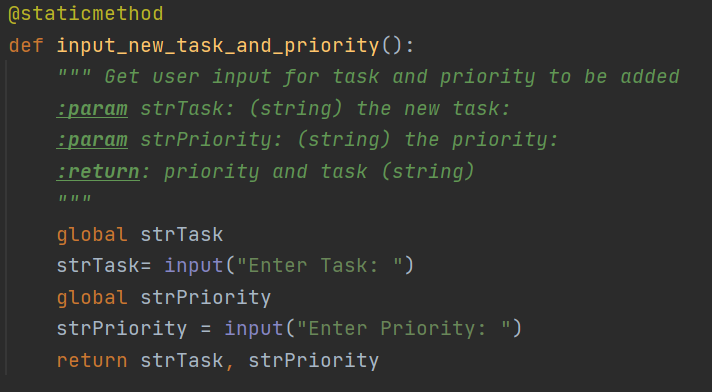
Robert F. Hiatt  
  
February 23, 2021  
  
IT FDN 110 B  
  
Assignment06  
  
[GitHub URL](https://github.com/roberthiatt/ITFDN110B-Mod06)

To-Do List Program in Python

1. Introduction  
This assignment builds upon Assignment 05 but now delves into user-made functions, parameters, doc strings, classes, and global and local variables. We’ll also create a GitHub Webpage with markdown language. As before, we’re writing a program to create and maintain a to-do list. With the use of self-programmed functions, though, the maintenance of our list becomes much easier.   
  
2. Using Someone Else’s Code  
Randal again provides the framework for this assignment. Download and open the file provided by Randal, Assigment06\_Starter.py. Read through the code to get a sense of what the program does. Amend the ChangeLog.  
  
  
***Script header with updated ChangeLog.***Randal provides comments and algorithms already. There are lines where Randal asks you to add data. Note that Randal uses what’s called *separation of concerns* in order to organize the script in a way that’s easier to read, discern, and potentially use later on for other endeavors.   
  
  
***Example of* separation of concerns *with the use of “Processing” in the comments.***  
  
Before we begin adding the requested code, let’s discuss some of the “new” material here. There are *functions*, which Randal provides, and these functions are similar to functions that are built into Python already, such as *range()*; the difference here is you get to create the functions and refer to them as needed. Next there are *classes*, which essentially is a group of functions (or variables, etc.). In the screenshot above, Randal provides us with the *class* called Processor. What follows will be the user-made *functions* within the *class*.

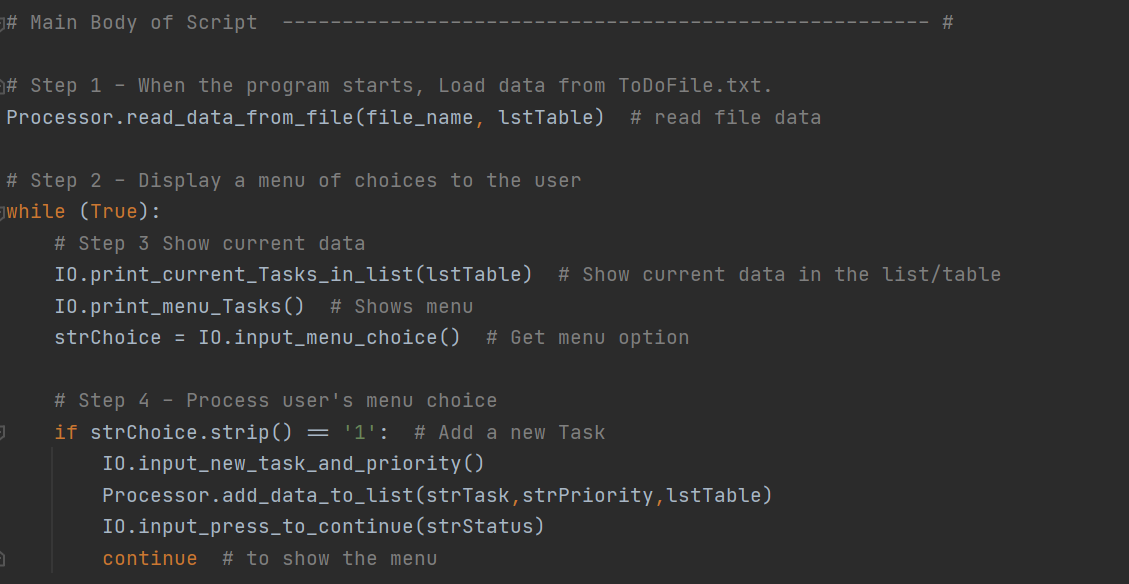
  
***The first* function *within the* class *Processor* *as indicated by the use of* “def.”**   
  
In the image above, the user-made function is “read\_data\_from\_file” and parenthetically there are *parameters*, which allow you to pass values into the function for processing. In this regard functions are useful in that they may be shorthand for larger blocks of code—using functions to call code is also referred to as *abstraction*. *Abstraction*, in general, is a short-hand way of processing a greater amount of information. In our textbook, the author uses the example of fast food workers using menu item numbers as abstraction (Dawson 162). Also of note in the above image is the use of *doc strings*, which help explain the function, its parameters, and other components. As it’s best practice to include *doc strings* for each function, I’ve done just that so that any future user can read and discern the code easily.

While the Processing section of our program creates a function for each menu option, the next section—abiding by the rule of *separation of concerns*—is concerned with input/output (IO).   
  
  
***This next* class *is called IO* *and the functions are related to input/output tasks.***

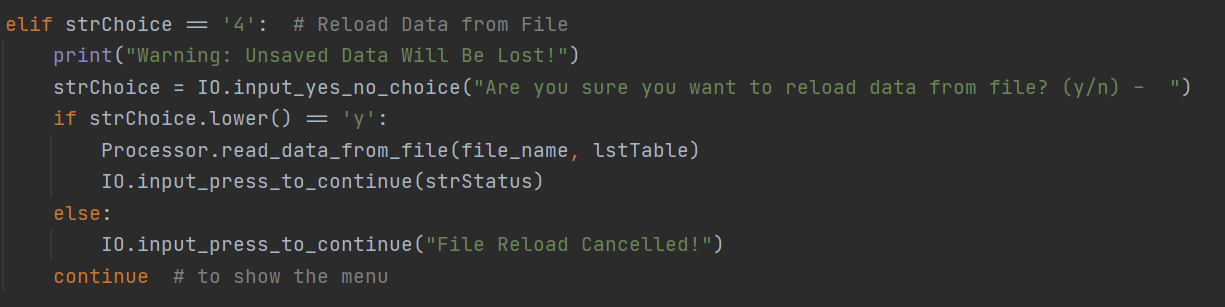
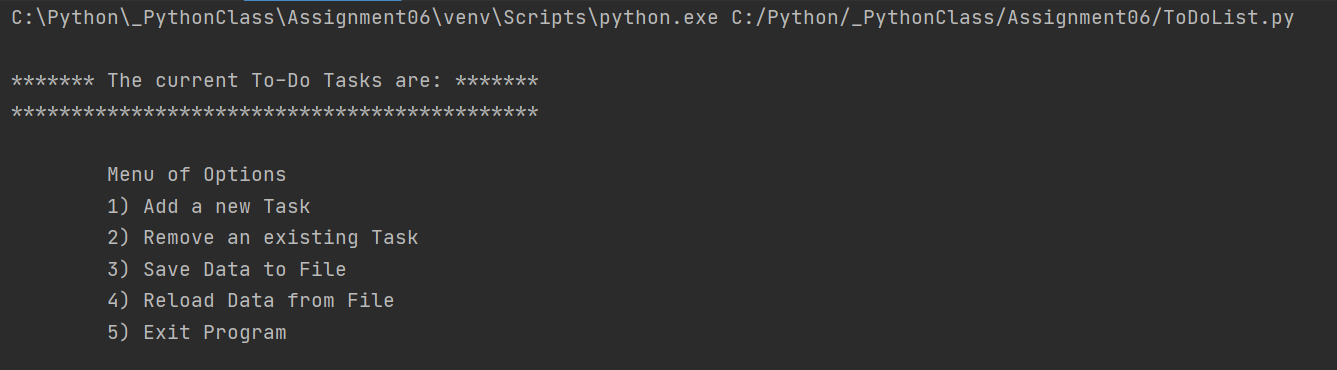
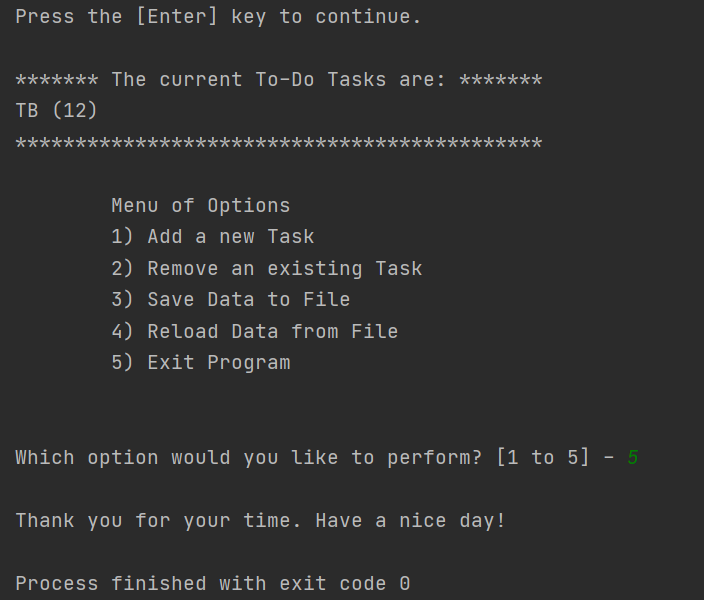
At a high level, the IO section follows the same lessons as the Processing section, and also involves material reviewed previously including nested functions and indexing. There are still some additional new items that Randal deploys.  
  


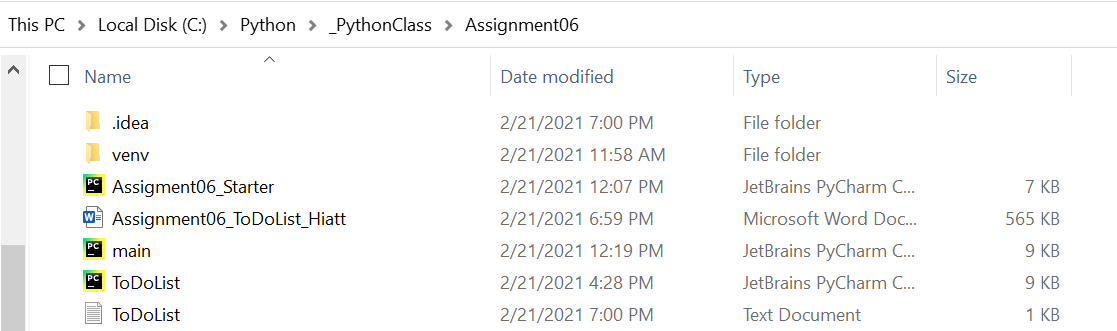
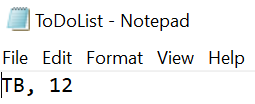
***Here we see the use of* global variables *and the use of* return *for variables*.**First let’s make the distinction between *local* and *global* variables. The local variables are specific to the functions where they’re written, which substantiates calling them *local*. And this concept is an extension of what’s referred to as *encapsulation*, which “helps keep independent code truly separate by hiding or encapsulating details” (Dawson 165).The difference, then, with *global* variables, is that you can transcend *scopes*, or, other areas of your program, by calling variables as *global*. The last bit of information to call out with respect to the above image is the use of the *return* function, which does just that—it *returns* variables and it also ends the function.

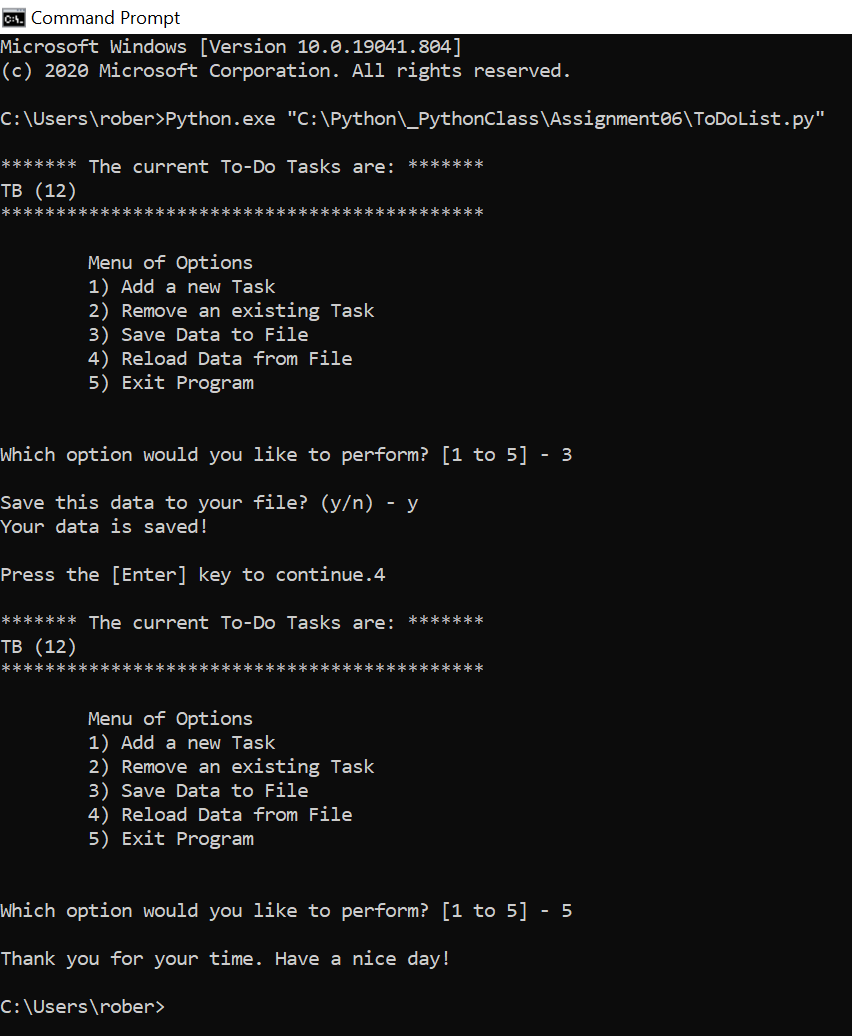
The program’s last section is entitled Main Body of Script.



***This section provides the payoff for the* classes *created previously. Here you can see that both the* Processor *and the* IO classes *are used with their respective functions called*.**   
The above image shows not only the value in created *classes*, it also demonstrates perfectly the idea of *abstraction*, wherein you can see the functions called that embody so many more lines of code. Here we efficiently execute the program.

In the Main Body of Script, we’re running through the program’s hypothetical user selections vis-à-vis a series of *If* statements.   
  
   
***The user may reload the data from the file or they may abort as seen through the* else *statement “File Reload Cancelled!”***  
  
One thing I want to call out that I really like is the communication in option 4. Here the user will be informed of their actions and it’s a nice touch to let them know explicitly how they’re using the program.  
With the new lessons explained for Assignment 06, let’s run the script in both PyCharm and the Command Line to ensure that the program is working properly.   
  
  
***Script goes to the menu in PyCharm***.  
  
  
***A run through the program, entering in a task and priority and then exiting the program with option 5.***

  
***The ToDoList.txt file saved properly.*** ***Here is the task and priority entered.***

Now let’s have a go on the Command Line:  
  
  
***The program ran successfully on the Command Line. In this image you can see options 3-5 working appropriately.***

The assignment’s final ask was to create a GitHub Webpage. Without recapitulating Randal’s instructions, I want to show you the end result after creating a new folder called docs and an index.md file. Once you’ve added the markdown language, your GitHub Webpage should look similar to this:  
  


3. Summary  
By this exercise’s completion you now have a program for the user to create and manage a to-do list. Distinct from Assignment 05, this new script is much more fluid and user-friendly. Further, you’ve gained an introduction and use case for defining functions, classes, the concepts of abstraction and encapsulation, and you’ve also created a GitHub Webpage!

Works Cited

Dawson, Michael. *Python Programming for the Absolute Beginner, Third Edition*. Course Technology, 2010.