Kevin Scales

Feb 23, 2021

Introduction to Programming (Python)

Assignment 6

<https://github.com/KevinLeighScales/-IntroToProg-Python-Mod06->

To Do List – Revisited

# Introduction

This assignment introduced working with functions and classes while continuing work with dictionaries and lists to add, delete, list, and load and save dictionary items of tasks and priorities. The data is string, with no check made that the priorities are in any particular format (1 to 10, high or low, etc.) using now-familiar loops and conditionals. The separation of concerns concept is clearly illuminated with functions and classes. The main body of the program reads much like a list of tasks to perform repeating in the order they are performed. As we shall see, the main body just loops through menu items with a large if-block, but almost every line is either a user-defined function that names its purpose, a built-in function like print, or a comment. Thus, the readability is enhanced, while the main downside is increased amounts of scrolling up and down.

# The Python Script

The first thing to note on the code is that the bulk of the lines aside from the main body, itself brief and streamlined as mentioned in the introduction, are used for two classes, called Processor and IO. These are given useful names. IO, input-output, is clearly for gathering and displaying information, while Processor is a tad more vague, but probably (without even reading it) is about actually manipulating the data by whatever is called for. In Processor, we see on reading that four functions are included, to read or write a file, and to add or remove data from our list. So Processor was aptly named.

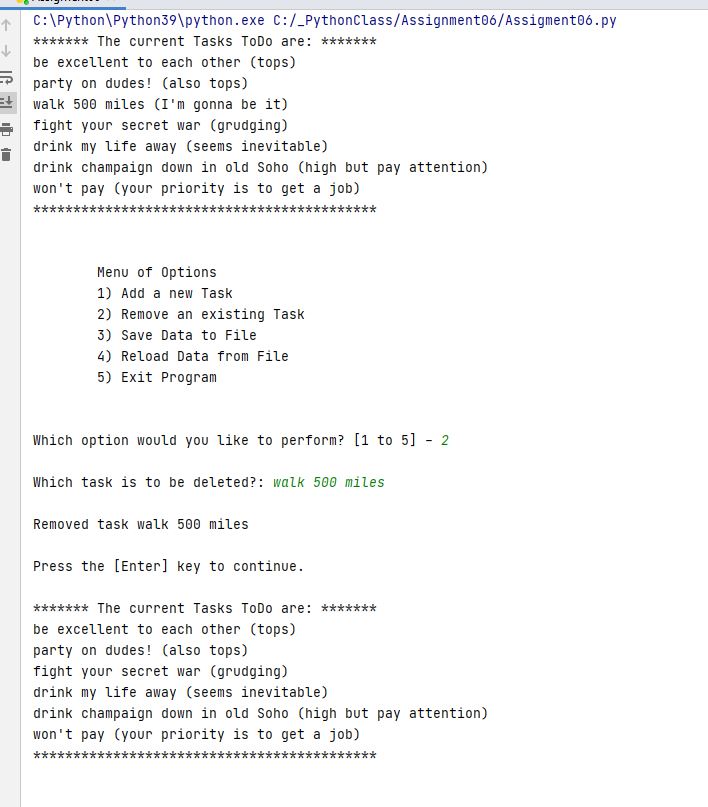
If we’ve paid super-close attention, we might note that the inputs to the functions, the arguments, have different names than the names passed in the main body section. This is no problem at all because passed variables are considered local once we are inside the function. So even though the variable in main lstTable is passed to Processor.add\_data\_to\_list, it is renamed list\_of\_rows upon arrival. This is neither right nor wrong, or even good or bad. Local variables are self-contained. Giving them identical names might help to remember them easier. Giving them similar but distinct names might emphasize that they are local. It is up to the coder. In this case it probably would have been fine to go with the same names (as a few are, like task and priority) because there are no global variables used in this program. A global doesn’t need to be passed to be available. It just needs to be identified as such. In that case, confusion with like-named variables could occur. Globals are sometimes frowned upon, and avoided much of the time for breaking up the isolation and separation of concerns pattern, so I tend to avoid them.

Continuing, the next class, IO, has within it seven functions, all including the word print or the word input. So our supposition that IO refers to sharing and collecting information seems to be verified. Some of these don’t even take arguments. Displaying the menu, for example, doesn’t need any new, changing information. It is a convenient way to block off a routine and often-repeated set of prints that, once written, probably won’t be revisited often. This one is noted also for returning nothing, but many functions actually provide some piece of information that is returned on completion. The task, the priority, the menu choice, a yes or no, or even the hitting of the enter key are all bits of data to be used after a function is called. In input\_new\_task\_and\_priority, we see that returning multiple data is not difficult. We just separate each by a comma (though lists and such can also be sent/returned).

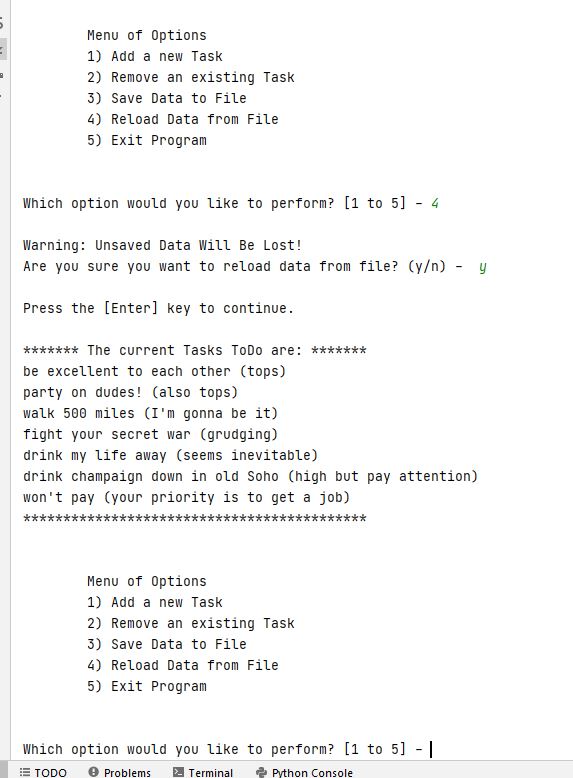
The rest of the code inside each function is largely unchanged from assignment 5 except to make our past code consistent with the starter code we began with. We can do inside a function pretty much anything that can be done in the main body.

# The code and output

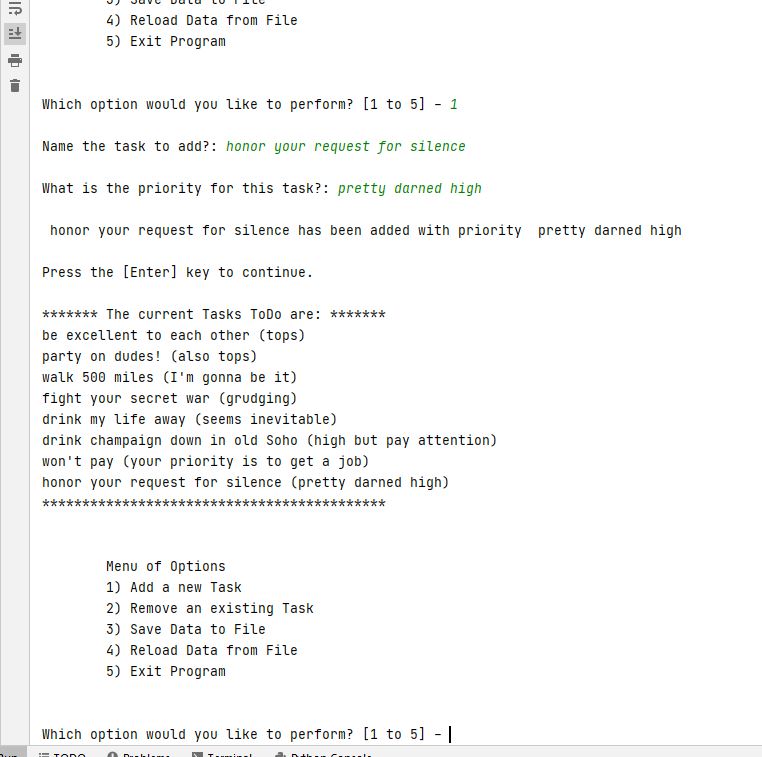
The code is attached in the zip file. Examples of the code working (not every single combination) are attached in the figures below.



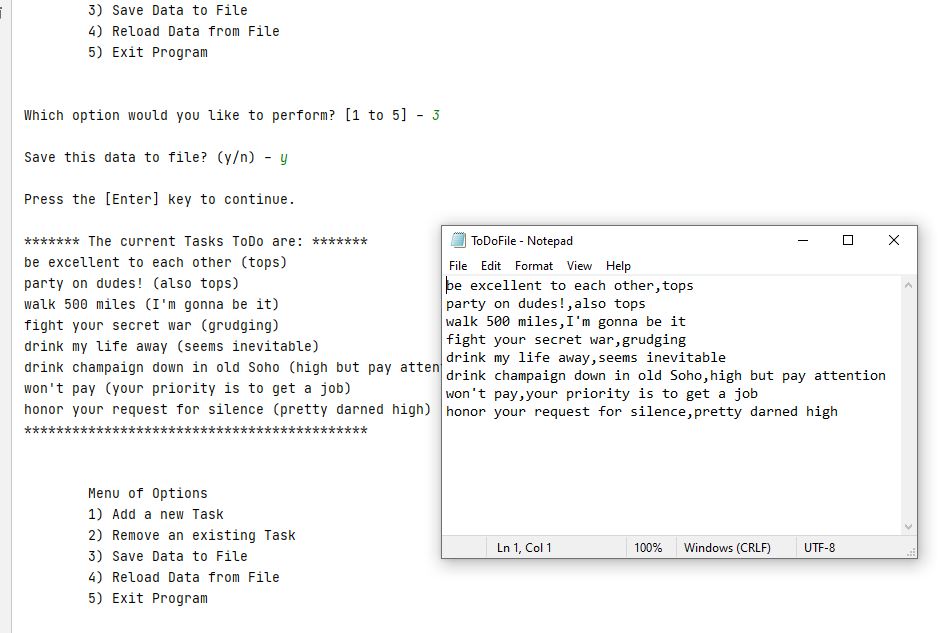
The process of removing an item from the list in memory is shown here.



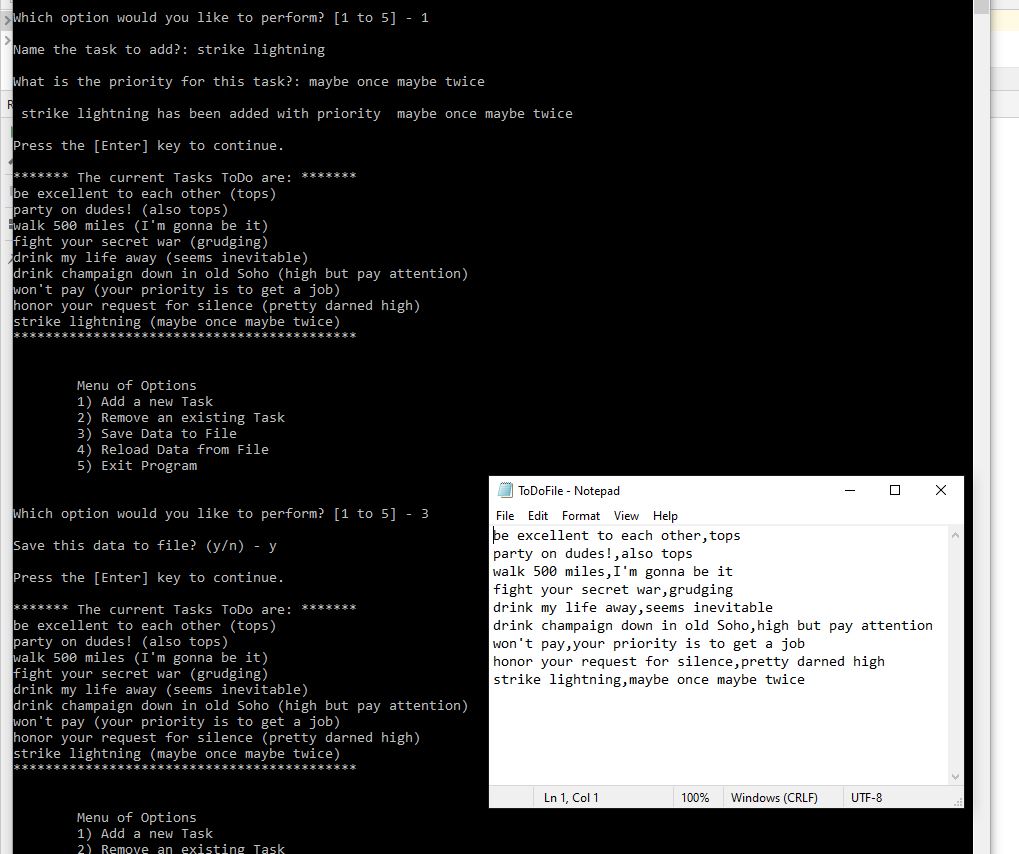
Reloading the file returns it to the state it was last saved in. Additions and removals are forgotten.



The task ‘walk 500 miles’ is back because we reloaded it. We add ‘honor your request for silence’ and its priority.



We can save it this way. If we run the program again, this will be our starting point. See below.



Now, in another console, we can add and save more stuff.

# Conclusion

We’ve seen functions streamline writing and understanding of code, and seen classes organize our functions. We don’t call a class per-se, but we call a function making reference to its class. We have passed arguments or parameters to functions, using the terms interchangeably for now, as is common, but this aspect may change in the next few assignments. Conversely, return values come back out of functions. Inside the functions, we work with local variables, passed in or created inside, but functions are kind of like Las Vegas. What happens in a function stays in a function (unless it is returned out). This is what local variables do. They don’t leave the function. They don’t do anything outside of the function. Global variables are available if we want to break this rules, but they are not considered good programming practice, so none are shown in this code.