

Thomas Nord

08-10-2022

Foundations of Programming (Python)

Recap of Module 06 – Functions & Classes

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

Creating a To Do List Program using Classes and Functions

Introduction

This week we're going to be looking at our To Do List program again, but this time we'll create/modify some custom classes and functions to use instead of re-writing entire blocks of code. This will help us avoid errors by reusing chunks of code rather than re-typing them when needed. We'll also look at how we can separate our processing layer from our presentation layer.

Getting Started

We start with the `Assignment06_starter_updated.py` file which has a good amount of code already in it. Our job this week is to figure out what is missing and write the code that will allow the program to function properly. The script uses 2 custom classes: `Processor()` and `IO()`. These classes represent our processing layer and our presentation layer, respectively. Inside each class are a number of custom functions written to carry out some bit of logic. We'll explore these in more detail as we go through the assignment. We also have 2 additional sections, or areas of concern: one for Data and the other for the main body of our script.

The variables we'll need are much like the ones from last week. We'll need to identify the name of the file we want to write to (`file_name_str`) as well as our file handle (`file_obj`). We'll be adding our data to dictionary rows and then those rows will be added to a list (`table_lst`). Finally, we'll capture the user's menu choice (`choice_str`).

Completing our Logic

My approach to completing the assignment is to walk through the logical flow of the program. I'll start with setting up the file then move on to displaying our menu, and then walking through the menu of options and ensuring each one is wired up correctly.

Creating the File/Reading the File's Data

The very first thing I want to do is borrow some code from Module 05 for creating a new document if one isn't already on disk. This is an incredibly helpful bit of code that saves time during testing as I can just delete the old file and programmatically generate a new one.

```
# Step 0 - When the program starts, if there is no file on disk, create one
if file_obj == None:
    file_obj = open(file_name_str, "a")
    file_obj.close()
```

In this conditional we evaluate the `file_obj` variable to see if it is set to `None` (i.e. there is no file already). If there isn't we open a new one in append mode which creates the file. We then immediately close the file as we will re-open it whenever we need to access it.

The code to read the existing data from the file is already in place so we don't need to do anything there. The code for the initial display of that data and to display our menu is also already in place so other than a few minor tweaks to the formatting we'll leave it alone.

Entering a New Task

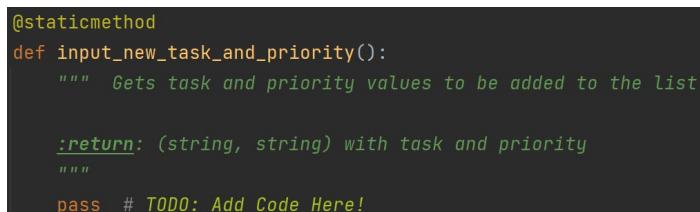
We then move on to entering a new task for our To Do List. This section of the main body of our script is in place but there is one change I need to make here. The parameters for the `Processor.add_data_to_list` function share names with our global variables, `task` and `priority`. As I feel this to be unnecessarily confusing, I modified them to add some clarity.

```
table_lst = Processor.add_data_to_list(task_to_add=task,
priority_to_add=priority, list_of_rows=table_lst)
```

Now when I see `task` in the code, I know exactly what the code is referring to. Don't forget to update your docstrings with the new names as well!

Tip: You can make quick work of the renaming by choosing the Refactor option in PyCharm.

If we scroll up to our functions in the script, we see we have our first bit of code to write (Figure 1).



```
@staticmethod
def input_new_task_and_priority():
    """ Gets task and priority values to be added to the list

    :return: (string, string) with task and priority
    """
    pass # TODO: Add Code Here!
```

Figure 1: Need code to capture user input

We're starting off with an easy one. We need code that takes no arguments but returns a user's input, in this case a task and its priority. We've covered capturing a user's input many times before so that is straightforward. However, now we have to return those captured values to the script so we add a `return` at the end followed by the name of our variables we assigned the user input to. Again, because we already have global variables for `task` and `priority` I name the variables inside the function something more specific to the action, `task_name` and `priority_level`.

Now we see when we select option 1 from the menu that we are asked to input a task and its priority. Once we do that our updated list of tasks is returned to us (Figure 2).

```
Which option would you like to perform? [1 to 4]: 1

Enter a task: Paint the House
Enter a priority: Low

Task added to list.
***** The current tasks ToDo are: *****
Paint the House (Low)
*****
```

Figure 2: Entering a task/priority

Deleting a Task

This section is going to require a bit more work. We have two functions and both are going to need code (Figure 3).

```
elif choice_str == '2': # Remove an existing Task
    task = IO.input_task_to_remove()
    table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
```

Figure 3: Two functions, both needing code

The first function we need to tackle is `input_task_to_remove` which receives no arguments, but will need to return a string for the task we want to remove (Figure 4).

```
@staticmethod
def input_task_to_remove():
    """ Gets the task name to be removed from the list

    :return: (string) with task
    """
    pass # TODO: Add Code Here!
```

Figure 4: Task removal function

This code should be very familiar as we're just assigning the user's input to a variable and returning that variable.

```
remove = str(input("Enter the name of the task you wish to remove:
")).strip()
return remove
```

Let's look at what we have already for the `Processor.remove_data_from_list` function (Figure 5).

```

@staticmethod
def remove_data_from_list(task, list_of_rows):
    """ Removes data from a list of dictionary rows

    :param task: (string) with name of task:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    # TODO: Add Code Here!
    return list_of_rows

```

Figure 5: Need to update variables and add logic

The first order of business for our removal code is to update the parameter for the function like we did earlier for adding to our list. Again, don't forget to you can use the Refactor option in PyCharm to update so only the local variables, but also the docstring.

```

@staticmethod
def remove_data_from_list(task_to_del, list_of_rows):

```

We see that we'll need to pass in the name of the task we want to delete (`task_to_del`) and our table (`list_of_rows`) then return an updated table (`list_of_rows`).

The next bit of code needs to evaluate the string in the `task_to_del` argument we pass to it to see if it matches any of the Tasks in our table. We'll do this with a `for` loop to read through each of the rows in our table and compare our `task_to_del` argument against each value in the Task keys of the dictionary rows. If we find a match we'll remove that row and confirm to the user that a task was removed.

```

for row in list_of_rows:
    if row["Task"].lower() == task_to_del.lower():
        list_of_rows.remove(row)
        print("Task removed.")

```

All that's left is to return an updated `list_of_rows` to the main script and present it to the user.

```

return list_of_rows

```

A quick test of our code reveals we are able to successfully remove erroneous rows (Figure 6).

```

***** The current tasks ToDo are: *****
Paint the House (Low)
Paint the Cat (High)
*****

Menu of Options
1) Add a New Task
2) Remove an Existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 2

Enter the name of the task you wish to remove: Paint the Cat
Task removed.
***** The current tasks ToDo are: *****
Paint the House (Low)
*****

```

Figure 6: We probably shouldn't paint the cat

Saving to a File

Saving our data to a file won't require any code changes to the main script so let's look at the `Processor.write_data_to_file` function (Figure 7).

```

@staticmethod
def write_data_to_file(file_name, list_of_rows):
    """ Writes data from a list of dictionary rows to a File

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    # TODO: Add Code Here!
    return list_of_rows

```

Figure 7: We've got some work to do

We can see from the existing parameters that we'll need to be able to pass in our file's name (`file_name`) and our table (`list_of_rows`). We'll also need to return our table (`list_of_rows`).

I want to start by displaying the current data that will be saved to the file. Luckily, we already have this code written elsewhere in the script under the `IO.output_current_tasks_in_list` function so I'll just borrow that.

```

print("***** The current items ToDo are: *****")
for row in list_of_rows:
    print(row["Task"] + "(" + row["Priority"] + ")")
print("*****")

```

No point in rewriting the wheel, er, script. This could probably have been its own function but we'll save that for another time. Next, we'll open our file in write mode. This code should be pretty familiar now as

we used something similar in Assignment 05. Of course, here we need to make sure we are using our local variables and that those variables don't exist outside the scope of this function.

```
for dicRow in list_of_rows:
    objFile.write(dicRow["Task"] + "," + dicRow["Priority"] + "\n")
```

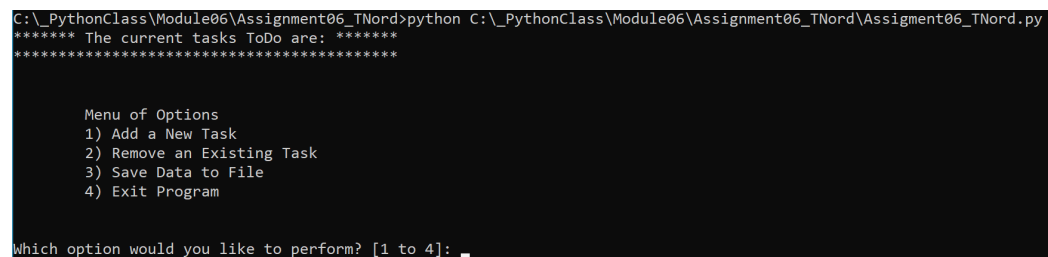
This will iterate through all the rows in our `list_of_rows` argument and write each one to the file. Don't forget to `close` the file when you're done! I'll add an input statement at the end to let the user know their data was saved and ask them to press Enter to return to the menu.

Exiting the Program

No new code was needed here.

Testing Our Program

We've seen our program running in PyCharm throughout its development. Let's fire up Command Prompt and make sure it is working there (Figure 8).

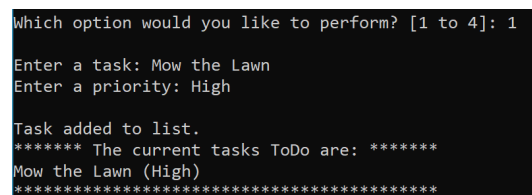


```
C:\_PythonClass\Module06\Assignment06_TNord>python C:\_PythonClass\Module06\Assignment06_TNord\Assignment06_TNord.py
***** The current tasks ToDo are: *****
*****
Menu of Options
1) Add a New Task
2) Remove an Existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: _
```

Figure 8: Our menu appears

We're off to a great start. Let's try the first option and see if we can add a new task and priority (Figure 9).



```
Which option would you like to perform? [1 to 4]: 1

Enter a task: Mow the Lawn
Enter a priority: High

Task added to list.
***** The current tasks ToDo are: *****
Mow the Lawn (High)
*****
```

Figure 9: Task and priority added

Oops! I misspelled one of my tasks. It's a good thing my removal code is working (Figure 10).

```

***** The current tasks ToDo are: *****
Mow the Lawn (High)
Wash the Car (Medium)
Wash the Car (Medium)
*****

Menu of Options
1) Add a New Task
2) Remove an Existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 2

Enter the name of the task you wish to remove: Wash the Car
Task removed.
***** The current tasks ToDo are: *****
Mow the Lawn (High)
Wash the Car (Medium)
*****

```

Figure 10: Delete successful

Now that we have a list of tasks let's save them off to a file and confirm the data is there (Figures 11.1 and 11.2).

```

Which option would you like to perform? [1 to 4]: 3
***** The current items ToDo are: *****
Mow the Lawn(High)
Wash the Car(Medium)
*****
Data saved to file! Press the [Enter] key to return to menu.
Data Saved!
***** The current tasks ToDo are: *****
Mow the Lawn (High)
Wash the Car (Medium)
*****

```

ToDoFile.txt - Notepad

File	Edit	View
Mow the Lawn,High		
Wash the Car,Medium		

Figures 11.1 and 11.2: Our data was saved and appears in the text file

One last test: can we properly exit the program (Figure 12).

```

Which option would you like to perform? [1 to 4]: 4

Goodbye!

C:\_PythonClass\Module06\Assignment06_TNord>

```

Figure 12: Goodbye!

Summary

This week we took another look at generating a to do list but using custom classes and functions instead of using only linear code. We also worked on further separating our code in to processing and presentation layers.