

Name: Cameron Allen

Date: May 18, 2023

Course: IT FDN 110 A

GitHub: <https://ct-allen.github.io/IntroToProg-Python-Mod06/>

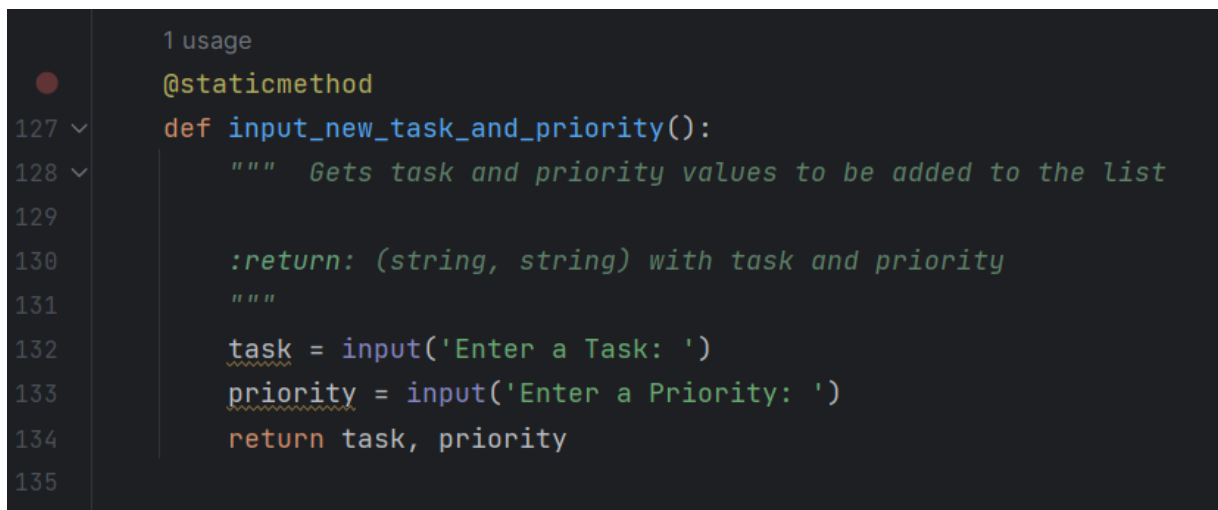
Assignment 06 – Functions

Intro

Our assignment this week was another test of working with existing code and adapting it with our own code. We introduce and start using 'classes' & 'functions'. These new features will enable us programmers to better organize our code, this is all part of the principle of 'Separation of Concerns'.

Input a Task

My first obstacle was getting 'input_new_task_and_priority' to work properly. It took some searching and to find that we are supposed to remove the 'pass' that was in this location. Then it took some more trial and error to determine that we must add the 'return' with the variable call outs inside it in order to get the user's inputs back to the main body code.



```
1 usage
  @staticmethod
127 def input_new_task_and_priority():
128     """ Gets task and priority values to be added to the list
129
130     :return: (string, string) with task and priority
131     """
132     task = input('Enter a Task: ')
133     priority = input('Enter a Priority: ')
134     return task, priority
135
```

Figure 1: First obstacle

Remove a Task

I wondered if you could use a 'return' statement within a 'if' statement to give back a value to the main code body to have it print a statement depending on what the 'Processor' class code did.

The simple way for now is to have the print statements in the 'Processor' code but this isn't quite in-line with the Separation of Concerns we are striving for.

```

57     @staticmethod
58     def remove_data_from_list(task, list_of_rows):
59         """ Removes data from a list of dictionary rows
60
61         :param task: (string) with name of task:
62         :param list_of_rows: (list) you want filled with file data:
63         :return: (list) of dictionary rows
64         """
65         for row in lstTable:
66             if row["Task"].lower() == strItem.lower():
67                 list_of_rows.remove(row)
68                 print("Row Removed")
69             else:
70                 print('Row Not Found')
71
72         return list_of_rows
73

```

Figure 2: Print statements within our 'Processor' code.

No File Found

At this point I have a full working program except for when a user doesn't have an existing ToDoFile.txt in their directory. So, to fix this I wanted to use the 'try' function list last week. At first, I was having errors but after fiddling around with it a bit it turned out to be a simple extra indentation that wasn't supposed to be there.

The screenshot shows a code editor with a Python function `read_data_from_file` and a console window displaying an error.

Code Editor:

```

30     :param file_name: (string) with name of file:
31     :param list_of_rows: (list) you want filled with file data:
32     :return: (list) of dictionary rows
33     """
34     try:
35         list_of_rows.clear() # clear current data
36         file = open(file_name, "r")
37         for line in file:
38             task, priority = line.split(",")
39             row = {"Task": task.strip(), "Priority": priority.strip()}
40             list_of_rows.append(row)
41         file.close()
42         return list_of_rows
43     except:
44         print('No ToDoFile.txt found, continue making list.')

```

Console Window:

```

Run Assignment06
C:\_PythonClass\Assignment06\venv\Scripts\python.exe C:\_PythonClass\Assignment06\Assignment06.py
File "C:\_PythonClass\Assignment06\Assignment06.py", line 34
    try:
IndentationError: unexpected indent
Process finished with exit code 1

```

Figure 3: An extra space just before the 'try' function that gave the error shown.

```
33         """
34         try:
35             list_of_rows.clear() # clear current data
36             file = open(file_name, "r")
37             for line in file:
38                 task, priority = line.split(",")
39                 row = {"Task": task.strip(), "Priority": priority.strip()}
40                 list_of_rows.append(row)
41             file.close()
42             return list_of_rows
43         except:
44             print('No ToDoFile.txt found, continue making list.')
45
Processor > read_data_from_file() > try
```

Run Assignment06 x

```
C:\_PythonClass\Assignment06\venv\Scripts\python.exe C:\_PythonClass\Assignment06\Assignment06.py
No ToDoFile.txt found, continue making list.
***** 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 4: Correct and working code.

```
Which option would you like to perform? [1 to 4] - 3

Data Saved!
***** The current tasks ToDo are: *****
Mow Lawn (low)
Homework (High)
dishes (med)
Haircut (low)
*****

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] - 4

Goodbye!
PS C:\_PythonClass\Assignment06>
```

Figure 5: Command Line verification

Summary

By adapting last weeks code with the week 6 code provided we made a new program that uses the principle 'Separation of Concerns'. With that we separate our code into different blocks, each block performs a different type of task. The blocks are called 'classes' and within each class we have different 'functions'. This is a very clean way to organize our code. However, when programming as a beginner it can be a little confusing with the functions calling other function which makes you have to jump around in your code and not just look at the next line.

After getting a grasp of the notes in the Assignment06_starter script and understanding the 'pass' and 'return' functions I was easily able to complete the program. I made sure to include code that would still let our program run even if the user didn't have a file called 'ToDoFile.txt' in their directory, which wasn't technically asked for in the prompt.