

# Creating a To-Do List Using Functions

## Introduction

Assignment 6 builds on the previous assignment where we created a To-Do List. This time though, instead of using loops, we use functions for gathering, processing, and displaying the data.

This script uses two classes: `Processor` and `IO`. The `Processor` class is used for data manipulation such as adding or deleting items from the list. This class is made up of four methods: `read_data_from_file`, `add_data_to_list`, `remove_data_from_list` and `write_data_to_file`.

The `IO` class is used for gathering information and displaying information back to the user. It consists of five methods: `output_menu_tasks`, `input_menu_choice`, `output_current_tasks_in_list`, `input_new_task_and_priority` and `input_task_to_remove`.

The main body of the script calls these functions depending on the menu choice selected by the user.

When the script begins it reads data from a text file. It does so by calling the “`read_data_from_file`” method of the `Processor` function. This is done in the background and is not apparent to the user.

```
## Step 1 - When the program starts, Load data from ToDoFile.txt.  
Processor.read_data_from_file(file_name=file_name_str, list_of_rows=table_lst) # read file data
```

### Listing 1

This function is expecting two arguments: `file_name` and `list_of_rows` as seen in **Listing 1** and **Listing 2**.

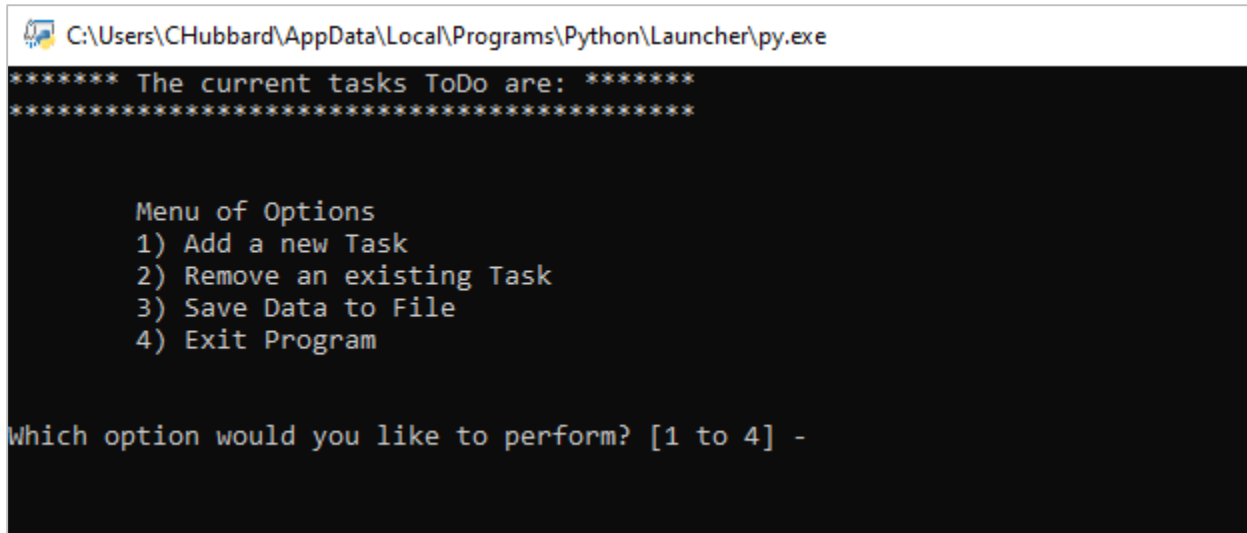
```
@staticmethod  
def read_data_from_file(file_name, list_of_rows):  
    """ Reads data from a file into a list of dictionary rows  
  
    :param file_name: (string) with name of file:  
    :param list_of_rows: (list) you want filled with file data:  
    :return: (list) of dictionary rows  
    """  
    list_of_rows.clear() # clear current data  
    file = open(file_name, "r")  
    for line in file:  
        task, priority = line.split(",")  
        row = {"Task": task.strip(), "Priority": priority.strip()}  
        list_of_rows.append(row)  
    file.close()  
    return list_of_rows
```

### Listing 2 The `read_data_from_file` method of the `Processor` class

You can see the `read_data_from_file` function in **Listing 2**. It opens the file in read-mode then loops through each row, separating the items into dictionary rows consisting of tasks and priorities. These rows are then assigned to a list, `table_lst`. When all rows in the file have been processed, the file is closed and the program returns to the main body of the script again.

## The Menu

When the script launches, the user is presented with a list of items currently on the To-Do List. The user is also presented with a list of menu options and asked to choose an option from the menu as seen in **Figure 1**.



```
C:\Users\CHubbard\AppData\Local\Programs\Python\Launcher\py.exe
***** 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 1** Blank To-Do list with menu options

This is accomplished by calling two methods of the IO class. First is the `output_current_tasks_in_lists` as seen here in **Listing 3**.

```
@staticmethod
def output_current_tasks_in_list(list_of_rows):
    """ Shows the current Tasks in the list of dictionaries rows

    :param list_of_rows: (list) of rows you want to display
    :return: nothing
    """

    if len(table_lst) == 0:
        print("*****There are", len(table_lst),"items on your To-Do List:*****")
    elif len(table_lst) < 2:
        print("*****There is 1 item on your To-Do list:*****")
    elif len(table_lst) > 1:
        print("***** There are", len(table_lst), "items on your ToDo list: *****")
    for i, item in enumerate(table_lst,1):
        print(i,"Task:", item["Task"].title() + " | " + "Priority:", item["Priority"].title())
    print("-"*54)
    print() # Add an extra line for looks
```

**Listing 3** `Output_current_tasks_in_lists` function

The second class is the `output_menu_tasks` function. The items in the To-Do list and the menu will be printed out each time a user selects options 1 – 3.

```
# Step 2 - Display a menu of choices to the user
while (True):
    # Step 3 Show current data
    IO.output_current_tasks_in_list(list_of_rows=table_lst) # Show current data in the list/table
    IO.output_menu_tasks() # Shows menu
    choice_str = IO.input_menu_choice() # Get menu option
```

#### **Listing 4 IO class and its methods**

The `output_menu_tasks` function takes no arguments. It simply prints out a text menu of options.

```
class IO:
    """ Performs Input and Output tasks """

    @staticmethod
    def output_menu_tasks():
        """ Display a menu of choices to the user

        :return: nothing
        """
        print('''
Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program
''')
        print() # Add an extra line for looks
```

#### **Listing 5 Code to display menu options**

Once the user makes a selection from the menu, the `input_menu_choice` function is called as seen in **Listing 5**.

```
@staticmethod
def input_menu_choice():
    """ Gets the menu choice from a user

    :return: string
    """
    choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()
    print() # Add an extra line for looks
    return choice
```

#### **Listing 6 Input\_menu\_choice function**

## Option 1 – Add Item to To-Do List

If the user selects **Option 1** to add a new task to the list, the `input_new_task_and_priority` function is called.

```
# Step 4 - Process user's menu choice
if choice_str.strip() == '1': # Add a new Task
    task, priority = IO.input_new_task_and_priority()
    table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
    continue # to show the menu
```

**Listing 7 Shows functions being called when user selects Option 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!
    strTask = str(input("What is the task? - ")).strip()
    strPriority = str(input("What is the priority? [high|low] - ")).strip()
    return strTask, strPriority
```

**Listing 8 Adding new task to the list**

The values returned are passed to the add\_data\_to\_list function.

```
@staticmethod
def add_data_to_list(task, priority, list_of_rows):
    """ Adds data to a list of dictionary rows

    :param task: (string) with name of task:
    :param priority: (string) with name of priority:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
    # TODO: Add Code Here!
    table_lst.append(row)
    print("Current Data in table:")
    return list_of_rows
```

**Listing 9 Returns user input for task and priority**

The new item is added to the list and the list contents are displayed to the users.

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

What is the task? - retire
What is the priority? [high|low] - high
Current Data in table:
***** The current tasks ToDo are: *****
win the lottery (high)
retire (high)
*****
```

**Figure 2 Adding item in command window**

```

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

What is the task? - Retire
What is the priority? [high|low] - high
Current Data in table:
***** The current tasks ToDo are: *****
win the lottery (high)
Retire (high)
*****

```

**Figure 3 Adding tasks in PyCharm**

## Option 2 – Remove Item from To-Do List

The user selects **Option 2** when they want to remove items from the To-Do list.

```

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)
    continue # to show the menu

```

**Listing 10**

The `input_task_to_remove` function is called and prompts the user for an item to be removed from the list.

```

@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!
    task = input("Which TASK would you like removed? - ")
    return task

```

**Listing 11 input\_task\_to\_remove function**

The value returned is then passed to the `remove_data_from_list` function where the item is removed from the `table_lst` list.

```

@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!
    for row in table_lst:
        if row["Task"].lower() == task.lower():
            table_lst.remove(row)
    return list_of_rows

```

**Listing 12** *remove\_data\_from\_list function*

The item has been removed and an updated list of items is presented to the user. You can see the item “retire” was removed from the list and now there is only one item remaining in the list.

```

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

Which TASK would you like removed? - retire
***** The current tasks ToDo are: *****
win the lottery (high)
*****

```

**Figure 4** *Removing item from command window*

```

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

Which TASK would you like removed? - Retire
***** The current tasks ToDo are: *****
win the lottery (high)
*****

```

**Figure 5** *Removing task in PyCharm*

## Option 3 – Save Data to File

If **Option 3** is selected, the data in `table_lst` is written to text file “ToDoFile.txt” This is done by calling the `write_data_to_file` function and passing in the name of the file to write to, along with the list that is to be written to the file. The user is notified that their data was written to the text file with the message “**Data Saved!**” displayed on the screen.

```

elif choice_str == '3': # Save Data to File
    table_lst = Processor.write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
    print("Data Saved!")
    continue # to show the menu

```

**Listing 13**



```

@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!
    objFile = open(file_name_str, "w")
    for item in table_lst:
        objFile.write(str(item["Task"]) + "," + str(item["Priority"]) + "\n")
    objFile.close()
    return list_of_rows

```

**Listing 14** *write\_data\_to\_file function*

```

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

Data Saved!
***** The current tasks ToDo are: *****
win the lottery (high)
*****

```

**Figure 6** *Data was saved to text file*

```

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

Data Saved!

```

**Figure 7**

## Option 4 – Exit the Program

If **Option 4** is selected, then the program exits. It calls the `input_task_to_quit` function which is just a simple input statement. Without this, when the script is run in a command window, the user never sees the “Goodbye!” printed to the screen. The window simply disappears. I felt this was a better approach.

```

elif choice_str == '4': # Exit Program
    print("Goodbye!")
    strExit = IO.input_task_to_quit()
    break # by exiting loop

```

### Listing 15

```
Which option would you like to perform? [1 to 4] - 4  
  
Goodbye!  
Press any key to exit
```

**Figure 8** Exiting script via PyCharm

```
Which option would you like to perform? [1 to 4] - 4  
  
Goodbye!  
Press any key to exit
```

**Figure 9** Exiting the scrip via command window

## Summary

This script demonstrates how to create a To-Do list using functions to gather, process and display data. The `Processor` class has several methods that are used to process the user-provided data and the `IO` class is used for gathering and displaying the data.