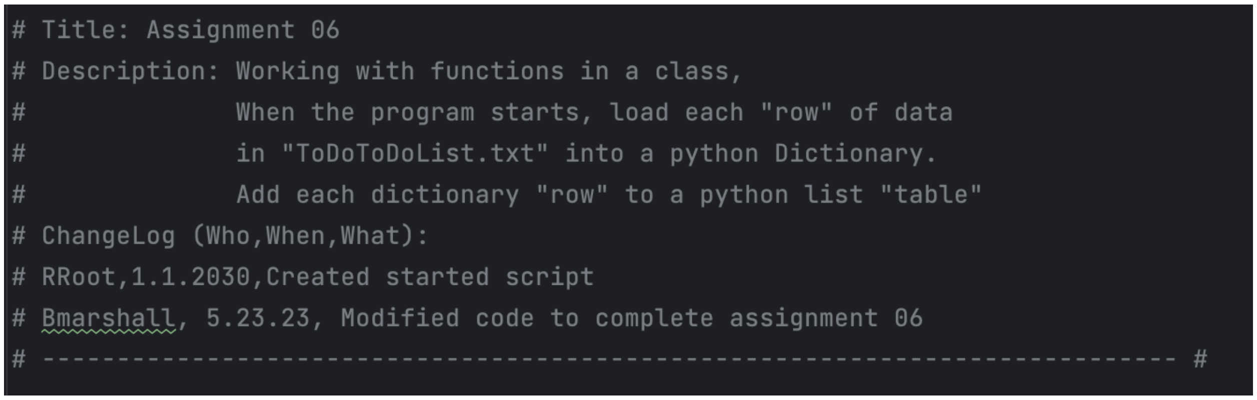
Bailey Marshall
May 22, 2023
IT FDN 110 A
Assignment 6
https://github.com/bmarshall14/IntroToProg-Python-Mod06/

Assignment 6: Using functions for more organized programming

Overview:
This week we learned about functions and how to use them to write more efficient programs. We have been using functions throughout this entire course such as print(), input(), remove(), etc. However, in addition to these pre-programmed functions, python allows you to create your own functions to perform specific tasks multiple times throughout a program. Functions are particularly important when writing long blocks of code in order to keep your script consistent and organized. To help organize functions, each function is grouped to its own class. While functions and classes can be a challenging concept to grasp, in the long run, they are helpful tools that will enable me to write more robust code in the long run.

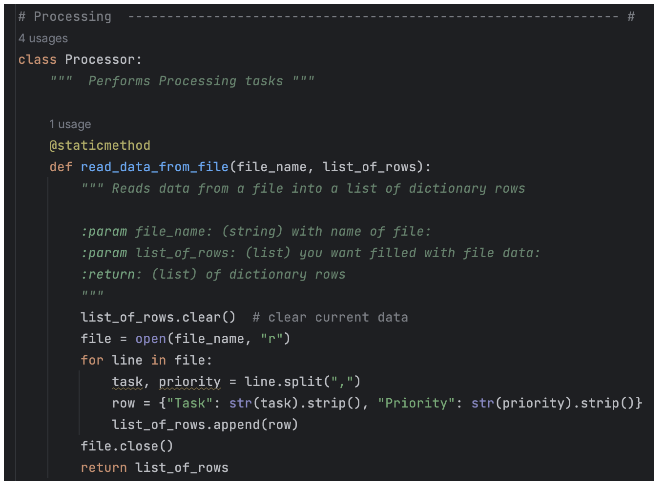
We also learned more about the debug feature in PyCharm. This feature is a built-in tool that allows you to run certain sections of your code in order to understand where bugs are coming from. This tool is especially useful when using functions, as functions are often repeated many times and can have various bugs in different sections of code.

Creating the script:
This week, the script is broken up into two classes and three sections of code called processor (class & section 1), presentation (class & section 2), and main body (section 3). I will describe the three sections, their associated functions and purpose, and the results below. Similar to last week, Dr. Root provided a template to start the code off of, and we were responsible for adding the code associated with each function. As this program involves making changes to a previously developed script, it is important to update the change log accordingly (Figure 1).

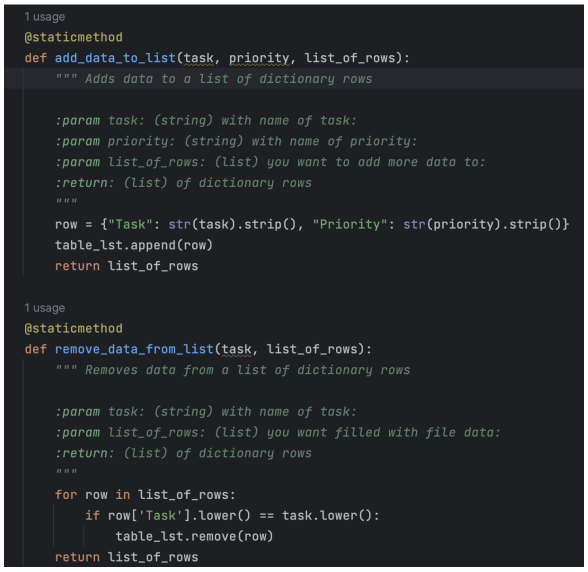

```
# Title: Assignment 06
# Description: Working with functions in a class,
#              When the program starts, load each "row" of data
#              in "ToDoToDoList.txt" into a python Dictionary.
#              Add each dictionary "row" to a python list "table"
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# Bmarshall, 5.23.23, Modified code to complete assignment 06
# ------------------------------------------------------------------------ #
```
Figure 1. Change log associated with Assignment 6

class:Processor

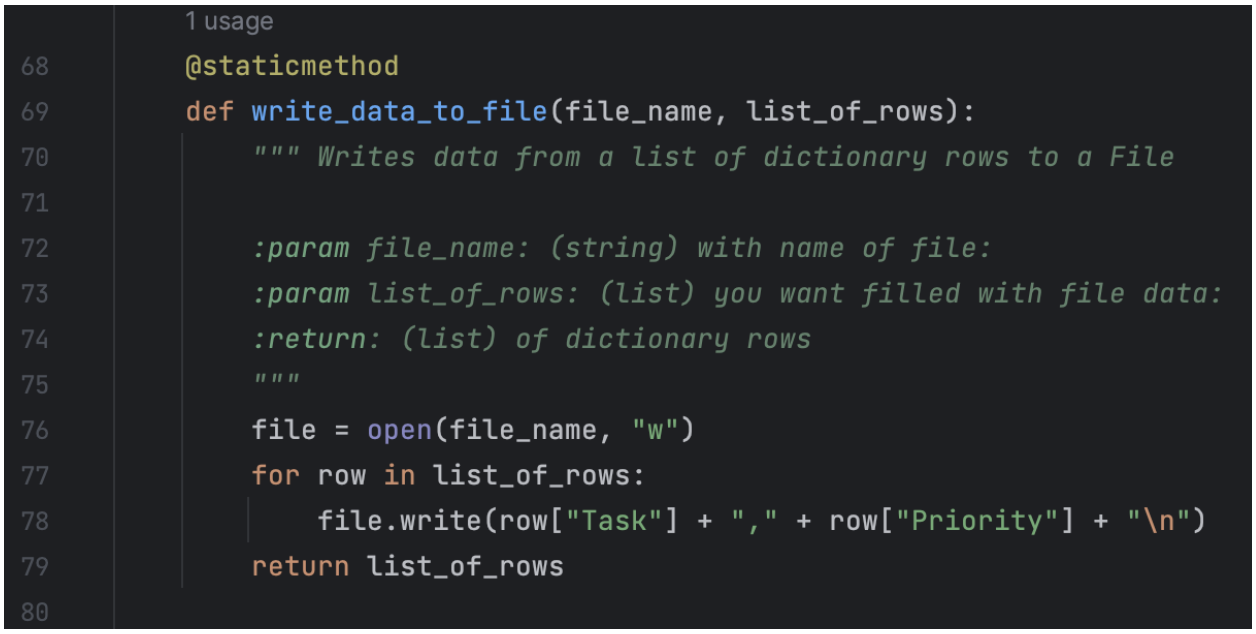
Unlike previous weeks where we included processing and presentation commands within the same block of code, this we separated out by class and function. The first class of functions is responsible for performing processing steps that are required to read and write data into a file that contains a list of dictionary rows. Once the file is loaded into the program, then functions were added to perform all of the actions associated with each menu option (Figure 2). No input or print functions are present within the block of code, as those functions are associated with the presentation class. When writing a new function, the command @staticmethod is used to define the function as a member of the class. In addition, it is important to use docstring, as defined by “””. Docstring serves as documentation for a class or function, and is critical when writing large programs to improve code readability.



```
# Processing  --------------------------------------------------------------- #
4 usages
class Processor:
    """  Performs Processing tasks """

    1 usage
    @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": str(task).strip(), "Priority": str(priority).strip()}
            list_of_rows.append(row)
        file.close()
        return list_of_rows
```



```
1 usage
@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 to add more data to:
    :return: (list) of dictionary rows
    """
    row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
    table_lst.append(row)
    return list_of_rows

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

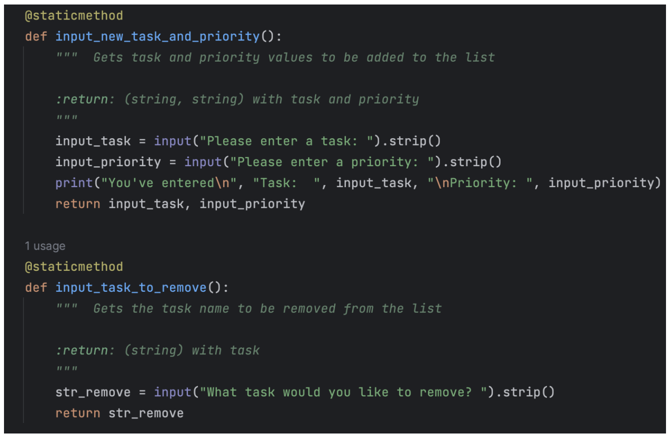


```
1 usage
@staticmethod
def write_data_to_file(file_name, list_of_rows):
    """ Writes data from a list of dictionary rows to a File

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    file = open(file_name, "w")
    for row in list_of_rows:
        file.write(row["Task"] + "," + row["Priority"] + "\n")
    return list_of_rows
```

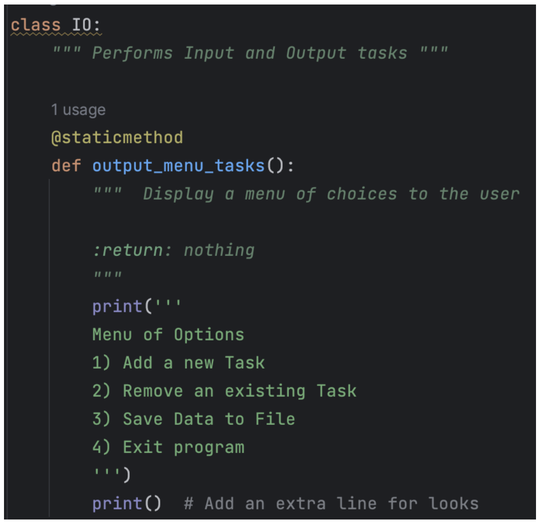
Figure 2. Code for all data processing steps associated with the To Do List program.

class:Presentation

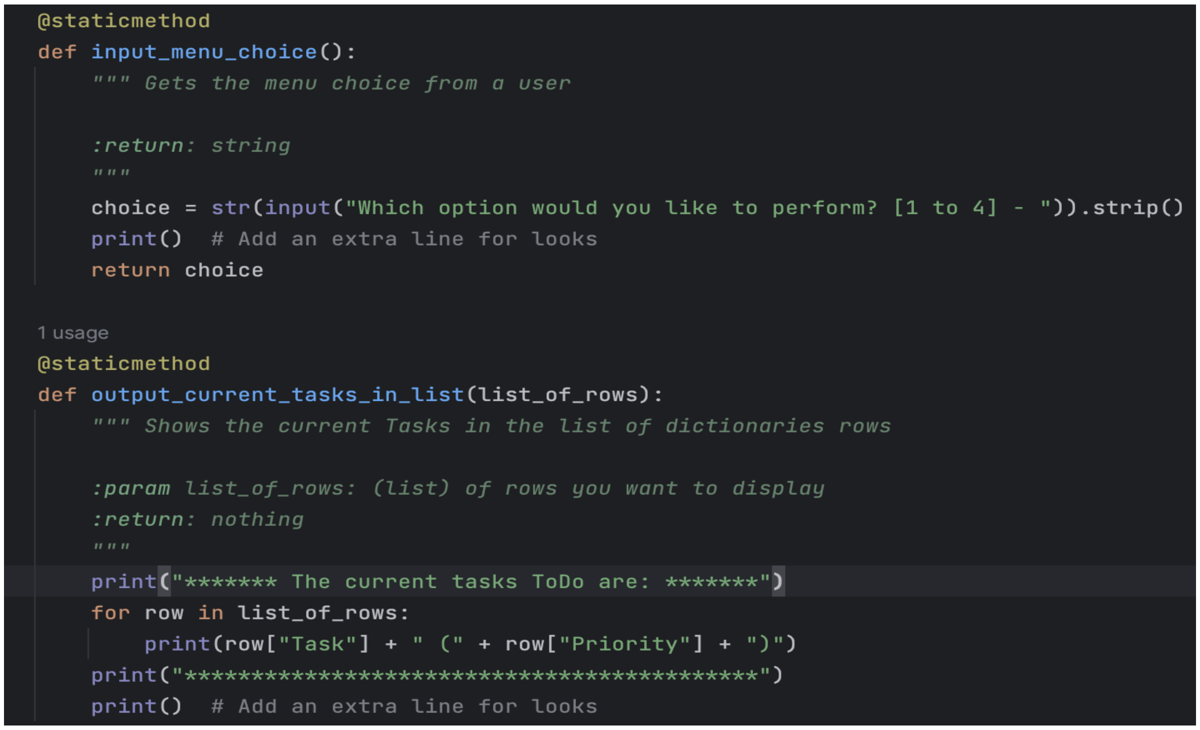
After defining what each function is responsible for, the next block of code focused on collecting and displaying information to the user. Similar to the first class, I began this section by defining a new class called IO. From there, all functions are associated with displaying the menu to the user, getting input data from the user to obtain a menu choice from the user, displaying the current list to the user, collecting a new task and priority form the user, and removing a task from the current list (Figure 3). When going through this section and reviewing the menu options, I was briefly confused where the function for option 3) Save data to file was. However, upon further thought, I remembered that the save data to file option does not display or obtain data from the user and instead, focuses on processing data. Because of this, the function associated with menu option 3 is in the processing class, rather than IO.



```
@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
    """
    input_task = input("Please enter a task: ").strip()
    input_priority = input("Please enter a priority: ").strip()
    print("You've entered\n", "Task:  ", input_task, "\nPriority: ", input_priority)
    return input_task, input_priority

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

    :return: (string) with task
    """
    str_remove = input("What task would you like to remove? ").strip()
    return str_remove
```



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

    1 usage
    @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
```

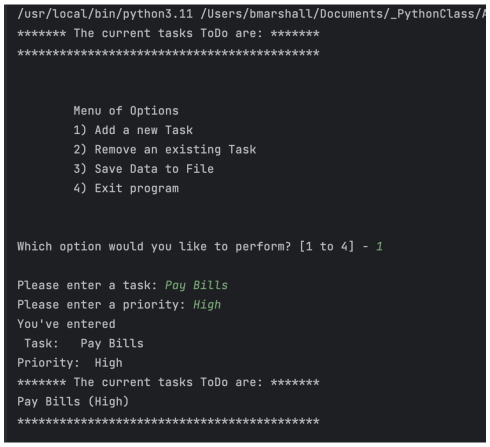


```
@staticmethod
def input_menu_choice():
    """ Gets the menu choice from a user

    :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

1 usage
@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
    """
    print("******* The current tasks ToDo are: *******")
    for row in list_of_rows:
        print(row["Task"] + " (" + row["Priority"] + ")")
    print("*******************************************")
    print()  # Add an extra line for looks
```

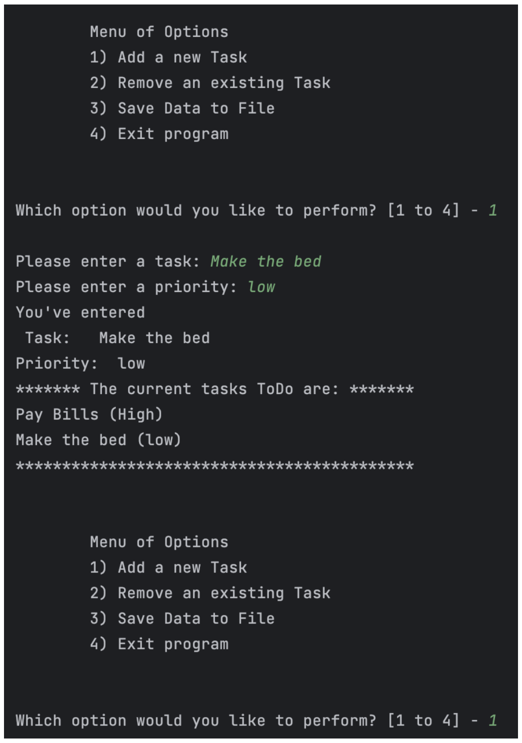
Figure 3. Code for all data presentation steps associated with the To Do List program.

Main body of script
After all the functions were defined, it was time to write the main body of the script which tells the program how and when to use each function. When calling a function that was defined earlier in the program, the format class.function is used. From there, any undefined variables are then defined for the specific function. For example, when loading data into the program from the ToDoFile.txt the command is as follows:
Processor.read_data_from_file(file_name=file_name_str, list_of_rows=table_lst). This command is telling the program to run the read_data_from_file function which is part of the processor class. In this function, both file_name and list_of_rows are undefined variables that allow for flexibility throughout the code. Because these are undefined, I then added in the file_name_str for file_name and tablelst for list_of_rows, which were declared earlier in the script. This format was carried about for each corresponding menu option. In addition, the main body of the script is where while loops can be implemented in order to control when and how functions run. In the case of this program, a while loop is used to display the menu options anytime menu options 1, 2, or 3 are selected from the user. If the user selects 1, 2, or 3, the function associated with the correct option will run, and the loop will continue back to the top to display the current to-do list and the menu options. The loop is broken when a user selects option 4 and the program ends (Figure 3).

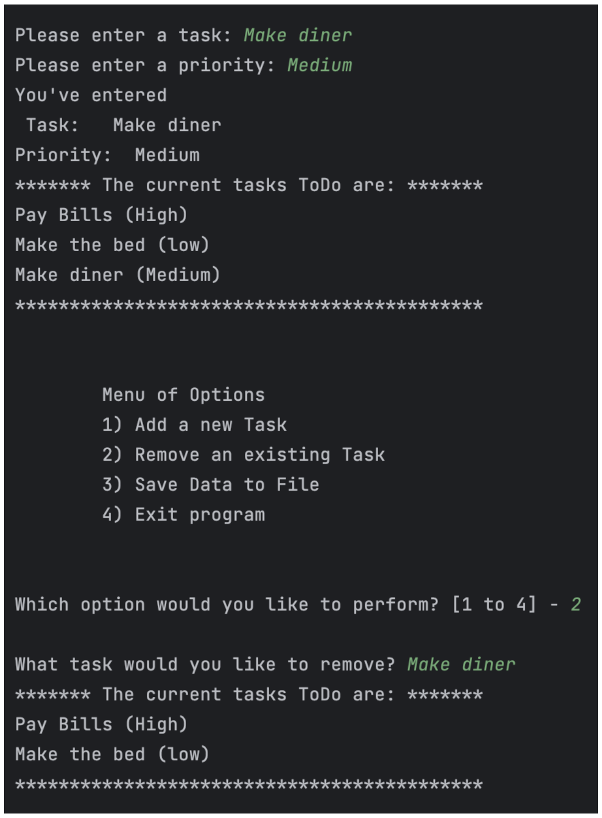
I then tested the code within the IDE, PyCharm, and the terminal. The code ran successfully in both cases, and an update text file named, ToDoFile.txt, was created in the working directory (Figure 4).



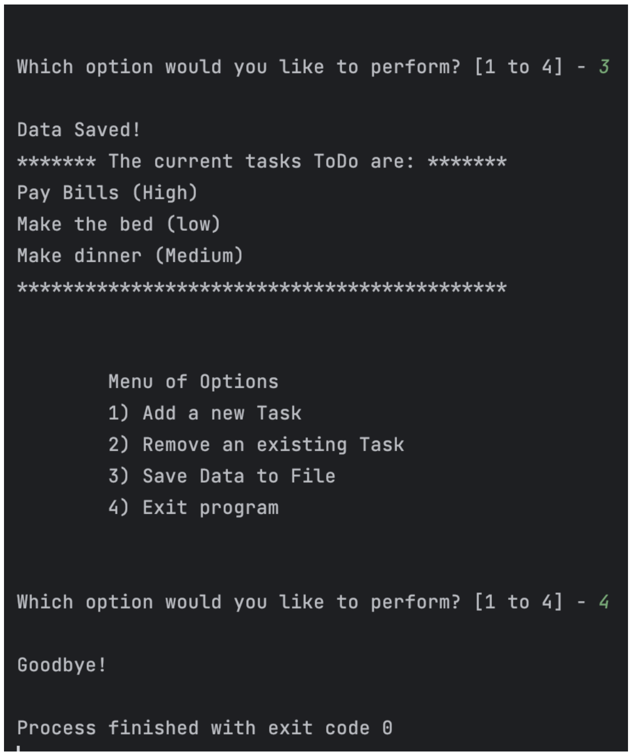
```
/usr/local/bin/python3.11 /Users/bmarshall/Documents/_PythonClass/
******* The current tasks ToDo are: *******
*******************************************


        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] - 1

Please enter a task: Pay Bills
Please enter a priority: High
You've entered
 Task:   Pay Bills
Priority:  High
******* The current tasks ToDo are: *******
Pay Bills (High)
*******************************************
```



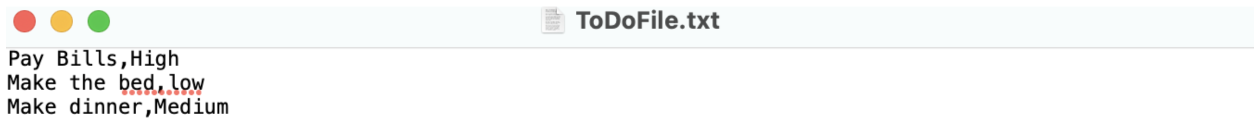
```
        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit program


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

Please enter a task: Make the bed
Please enter a priority: low
You've entered
 Task:   Make the bed
Priority:  low
******* The current tasks ToDo are: *******
Pay Bills (High)
Make the bed (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] - 1
```



```
Please enter a task: Make diner
Please enter a priority: Medium
You've entered
 Task:   Make diner
Priority:  Medium
******* The current tasks ToDo are: *******
Pay Bills (High)
Make the bed (low)
Make diner (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

What task would you like to remove? Make diner
******* The current tasks ToDo are: *******
Pay Bills (High)
Make the bed (low)
*******************************************
```



```

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

Data Saved!
******* The current tasks ToDo are: *******
Pay Bills (High)
Make the bed (low)
Make dinner (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] - 4

Goodbye!

Process finished with exit code 0
```



ToDoFile.txt

```
Pay Bills,High
Make the bed,low
Make dinner,Medium
```

Figure 4. Results from testing ToDoList.py

Summary:
This week, we utilized functions to perform a similar task as last week's To Do List program. While difficult to learn, functions are an important tool when developing large and complex programs. Functions offer many benefits like increasing organization and readability, writing more efficient code, and the ability to debug smaller sections of code. However, regardless of how organized you design your code, programs are inevitable. Handy tools like debug are available within IDEs like PyCharm and can also be utilized to help overcome some of the challenges associated with creating your own functions.