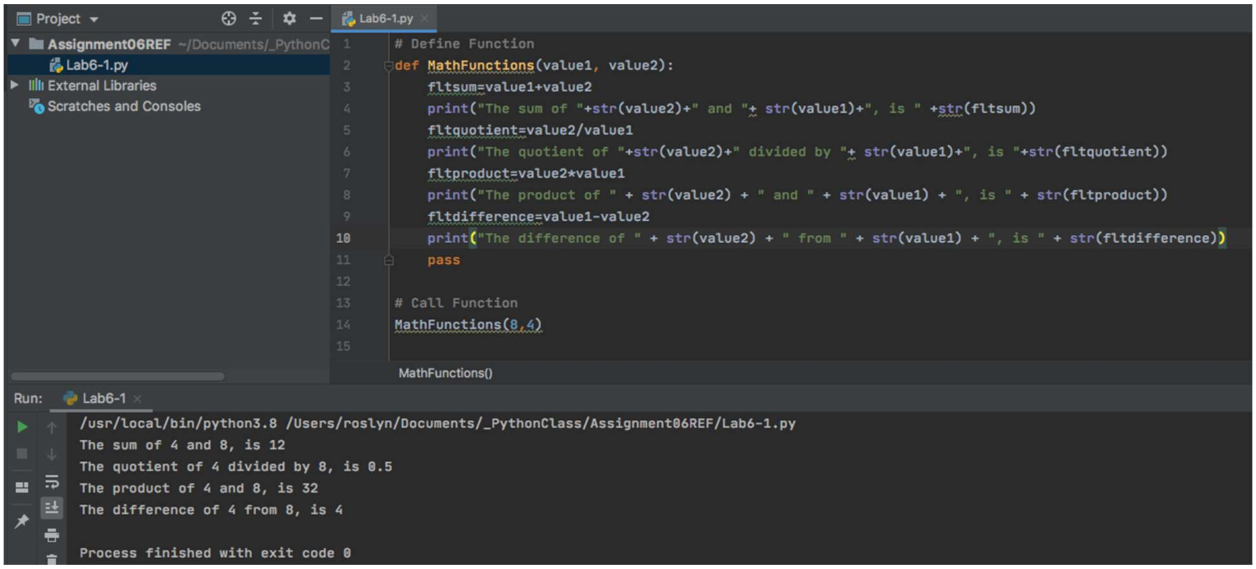
Roslyn Melookaran
5/22/2020
Foundations of Programming, Python
Assignment 6
https://github.com/roslynm/IntroToProg-Python-Mod06

Intro

In this module we learn how to use functions, and use classes to organize them. Functions allow us to better organize our script, and make it easy for trouble shooting, and to be picked up by downstream users. We gain more experience working with someone else's code and modifying it. We also get to see firsthand how much more organized a code can be with functions by comparing our module 6 scripts with module 5's. The following document is notes taken during this week's lecture, as well as a breakdown of the module 6 assignment.

Functions

Functions are a set of statements, grouped together under a given name. These functions are defined, and then called upon in the main program. Using functions in your program can make your program easier to read, and enables you to use the same function, in multiple scripts. Function can be used to process values, or parameters. Usually, in python, when working with parameters, you do not add prefixes that define the type of parameter (int/str/flt) within the function. See **Figure 1** for an example of a simple function.

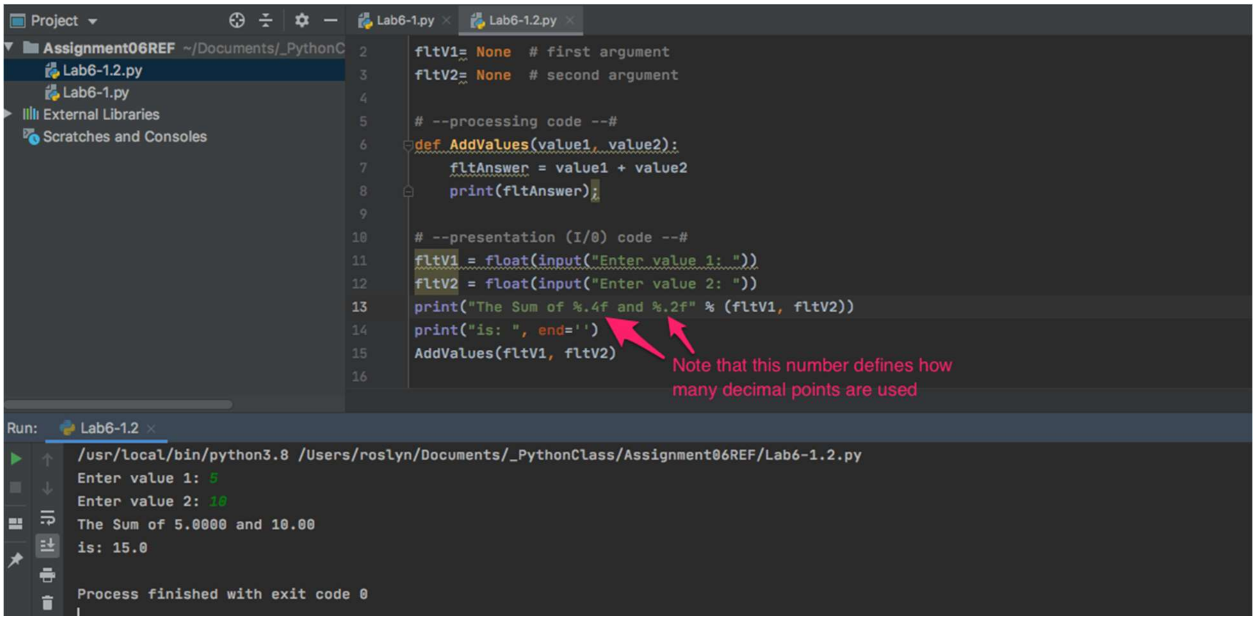


```python
# Define Function
def MathFunctions(value1, value2):
    fltsum=value1+value2
    print("The sum of "+str(value2)+" and "+ str(value1)+", is " +str(fltsum))
    fltquotient=value2/value1
    print("The quotient of "+str(value2)+" divided by "+ str(value1)+", is "+str(fltquotient))
    fltproduct=value2*value1
    print("The product of " + str(value2) + " and " + str(value1) + ", is " + str(fltproduct))
    fltdifference=value1-value2
    print("The difference of " + str(value2) + " from " + str(value1) + ", is " + str(fltdifference))
    pass

# Call Function
MathFunctions(8,4)
```

```
/usr/local/bin/python3.8 /Users/roslyn/Documents/_PythonClass/Assignment06REF/Lab6-1.py
The sum of 4 and 8, is 12
The quotient of 4 divided by 8, is 0.5
The product of 4 and 8, is 32
The difference of 4 from 8, is 4

Process finished with exit code 0
```

Figure 1. Simple function example

In **Figure 2**, you see another example of using a function. Note the difference in how we present this data using substitutions instead of concatenating strings like **Figure 1**.



```
fltV1= None  # first argument
fltV2= None  # second argument

# --processing code --#
def AddValues(value1, value2):
    fltAnswer = value1 + value2
    print(fltAnswer);

# --presentation (I/0) code --#
fltV1 = float(input("Enter value 1: "))
fltV2 = float(input("Enter value 2: "))
print("The Sum of %.4f and %.2f" % (fltV1, fltV2))
print("is: ", end='')
AddValues(fltV1, fltV2)
```

Note that this number defines how many decimal points are used

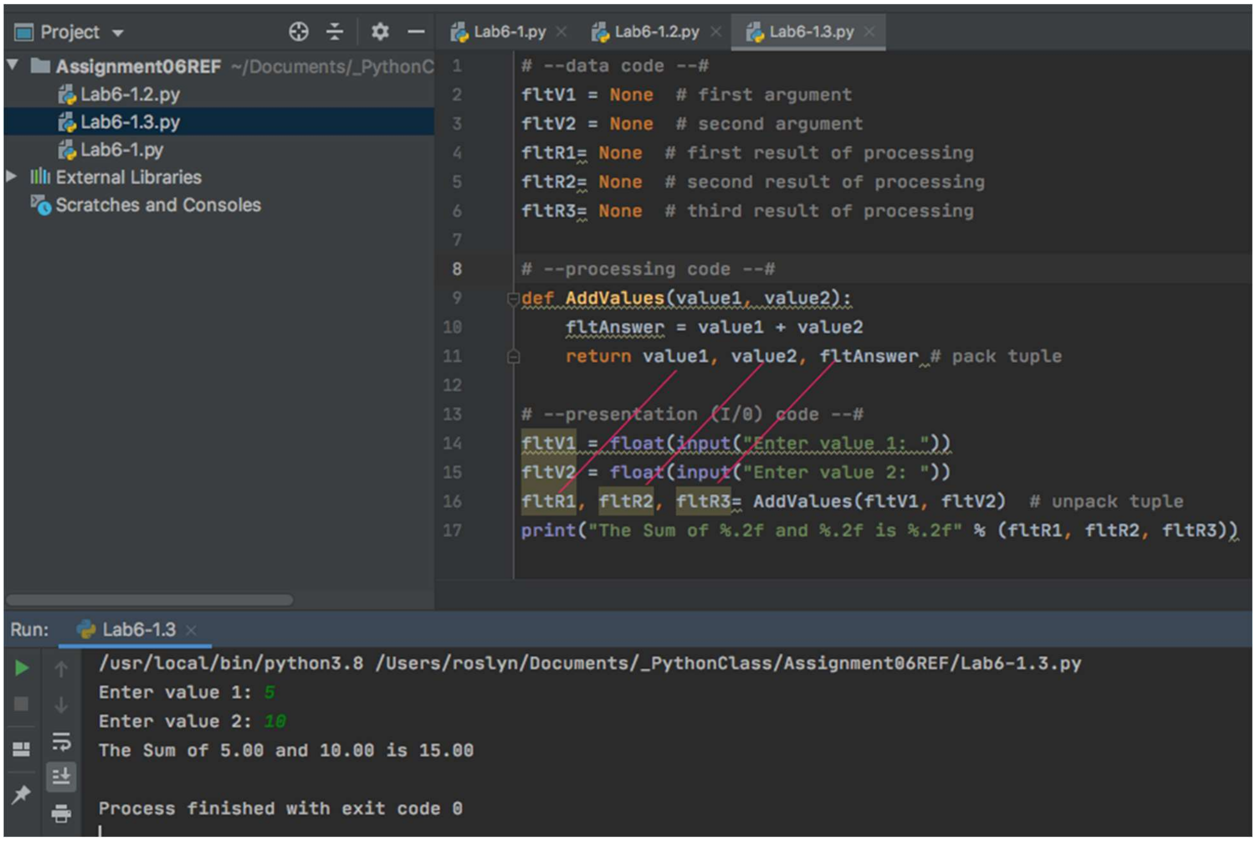
```
/usr/local/bin/python3.8 /Users/roslyn/Documents/_PythonClass/Assignment06REF/Lab6-1.2.py
Enter value 1: 5
Enter value 2: 10
The Sum of 5.0000 and 10.00
is: 15.0

Process finished with exit code 0
```

Figure 2. Example using functions

Return Values

After a function is called, you can use that processing in multiple ways. If you take the processing data and put it into a variable, it can be used again many times throughout your script. To do this, you would use a "return" line at the end of your function (e.g. return fltanswer). You can also use your function directly, as an expression. This means using the value of a function immediately, without passing it into a variable. An example of this would be passing it directly into a print function (e.g. print(MyFunction()). When using a function as an expression, it cannot be called upon later in the script, it is a one and done use.

You can also return values from a function in touple form. Note when you use the return function, with a comma separated list, it is an assumed touple. See **Figure 3** example below:

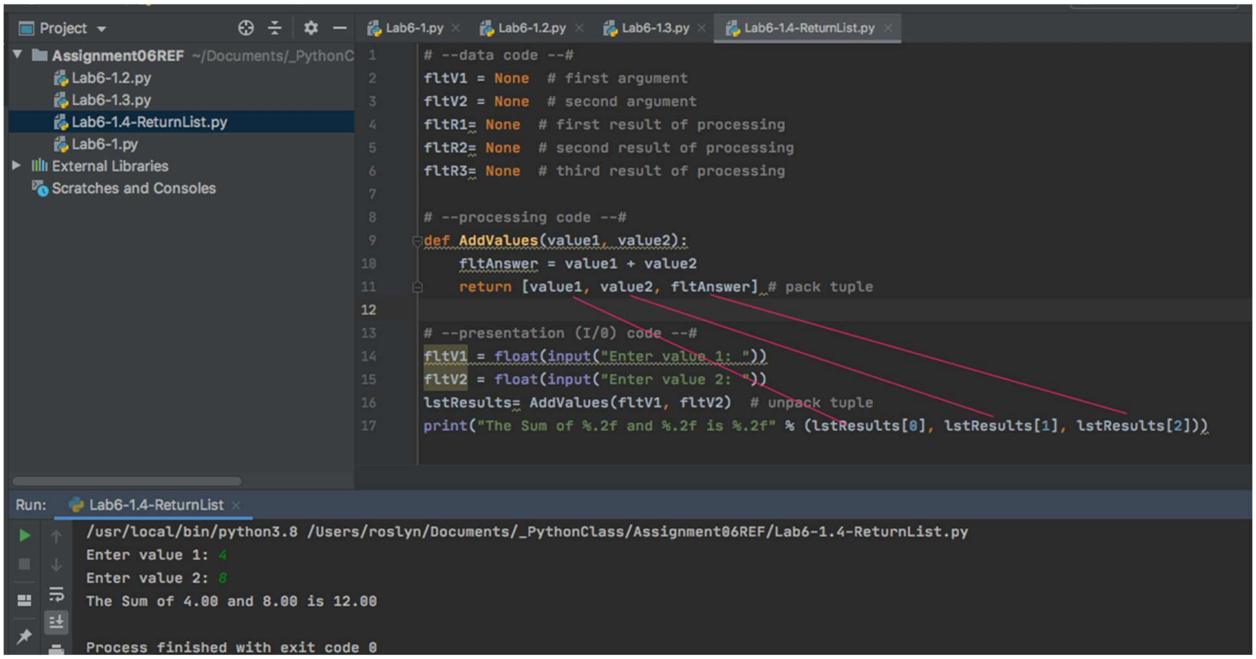


```python
# --data code --#
fltV1 = None  # first argument
fltV2 = None  # second argument
fltR1= None  # first result of processing
fltR2= None  # second result of processing
fltR3= None  # third result of processing

# --processing code --#
def AddValues(value1, value2):
    fltAnswer = value1 + value2
    return value1, value2, fltAnswer # pack tuple

# --presentation (I/0) code --#
fltV1 = float(input("Enter value 1: "))
fltV2 = float(input("Enter value 2: "))
fltR1, fltR2, fltR3= AddValues(fltV1, fltV2)  # unpack tuple
print("The Sum of %.2f and %.2f is %.2f" % (fltR1, fltR2, fltR3))
```

```
/usr/local/bin/python3.8 /Users/roslyn/Documents/_PythonClass/Assignment06REF/Lab6-1.3.py
Enter value 1: 5
Enter value 2: 10
The Sum of 5.00 and 10.00 is 15.00

Process finished with exit code 0
```

Figure 3. Returning a function as a touple

You can also return a list from a function too. However, in python touples are more commonly used to return functions. **Figure 4** shows an example returning a list from a function.



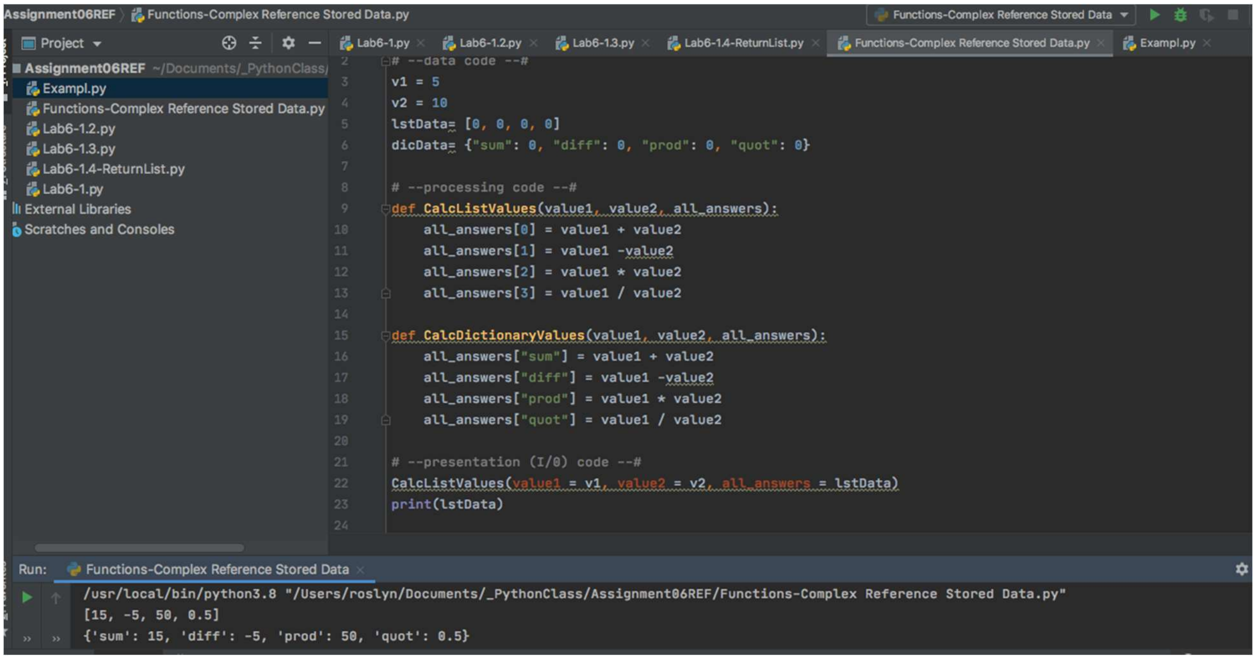
```python
# --data code --#
fltV1 = None  # first argument
fltV2 = None  # second argument
fltR1= None  # first result of processing
fltR2= None  # second result of processing
fltR3= None  # third result of processing

# --processing code --#
def AddValues(value1, value2):
    fltAnswer = value1 + value2
    return [value1, value2, fltAnswer] # pack tuple

# --presentation (I/0) code --#
fltV1 = float(input("Enter value 1: "))
fltV2 = float(input("Enter value 2: "))
lstResults= AddValues(fltV1, fltV2)  # unpack tuple
print("The Sum of %.2f and %.2f is %.2f" % (lstResults[0], lstResults[1], lstResults[2]))
```

```
/usr/local/bin/python3.8 /Users/roslyn/Documents/_PythonClass/Assignment06REF/Lab6-1.4-ReturnList.py
Enter value 1: 4
Enter value 2: 8
The Sum of 4.00 and 8.00 is 12.00

Process finished with exit code 0
```

Figure 4. Returning a list from function

You can create multiple versions of a function. To do this, you pass different numbers of parameters, of parameters with different data into them. When a function is used multiple times, it is called an overloaded function.

It is also important to keep track of whether data passing through functions are stored as values, or reference types. In **Figure 5**, you can see that a touple would not work here, because the function requires the difference indexes of the list to change. Since touples are immutable, this change would result in an error.


Assignment06REF
Functions-Complex Reference Stored Data.py
Functions-Complex Reference Stored Data
Project
Lab6-1.py
Lab6-1.2.py
Lab6-1.3.py
Lab6-1.4-ReturnList.py
Functions-Complex Reference Stored Data.py
Exampl.py
Assignment06REF ~/Documents/_PythonClass/
Exampl.py
Functions-Complex Reference Stored Data.py
Lab6-1.2.py
Lab6-1.3.py
Lab6-1.4-ReturnList.py
Lab6-1.py
External Libraries
Scratches and Consoles
Run:
Functions-Complex Reference Stored Data

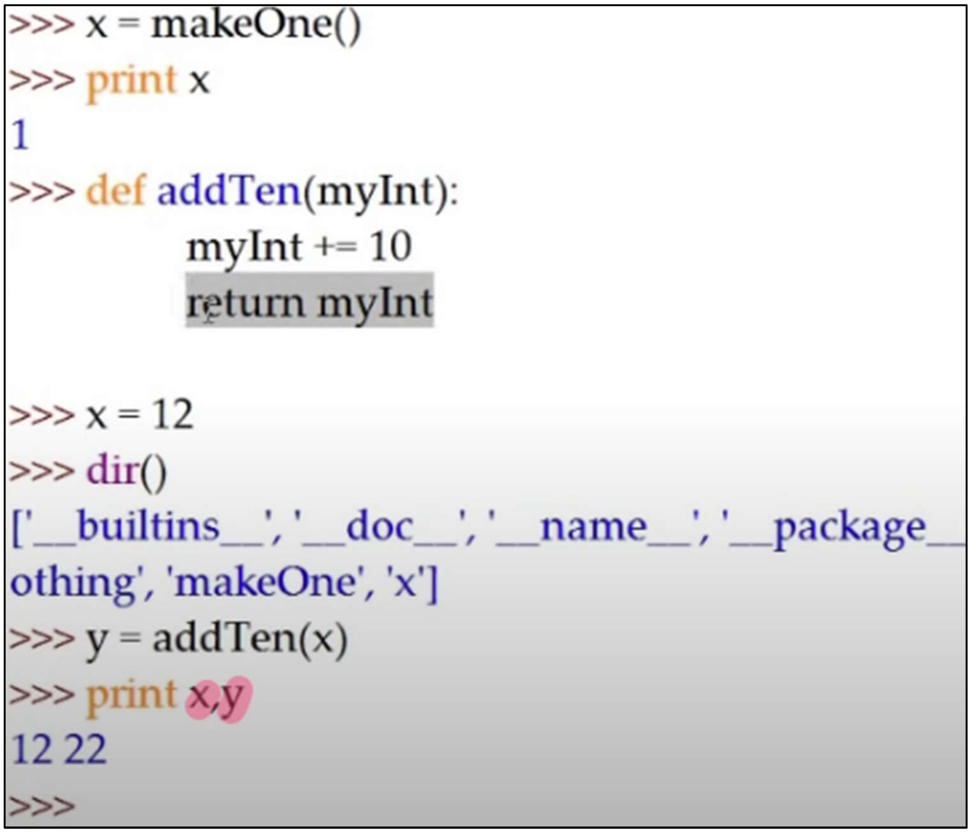
```
# --data code --#
v1 = 5
v2 = 10
lstData= [0, 0, 0, 0]
dicData= {"sum": 0, "diff": 0, "prod": 0, "quot": 0}

# --processing code --#
def CalcListValues(value1, value2, all_answers):
    all_answers[0] = value1 + value2
    all_answers[1] = value1 -value2
    all_answers[2] = value1 * value2
    all_answers[3] = value1 / value2

def CalcDictionaryValues(value1, value2, all_answers):
    all_answers["sum"] = value1 + value2
    all_answers["diff"] = value1 -value2
    all_answers["prod"] = value1 * value2
    all_answers["quot"] = value1 / value2

# --presentation (I/0) code --#
CalcListValues(value1 = v1, value2 = v2, all_answers = lstData)
print(lstData)
```

```
/usr/local/bin/python3.8 "/Users/roslyn/Documents/_PythonClass/Assignment06REF/Functions-Complex Reference Stored Data.py"
[15, -5, 50, 0.5]
{'sum': 15, 'diff': -5, 'prod': 50, 'quot': 0.5}
```

Figure 5. Example of using complex data (stored as reference type) in functions

Global Vs Local Variables

Variables that are declared in the function, and used only in the functions are called local variables. When variables are called on outside of the functions, they are called Global variables and are usually defined in the in the top of the script. You cannot call a local variable from a function outside the function. Using global variables within a function is not considered a best practice. In **Figure 6**, from (https://youtu.be/qO4ZN5uZSVg) , you can see how global and local variables make a difference.



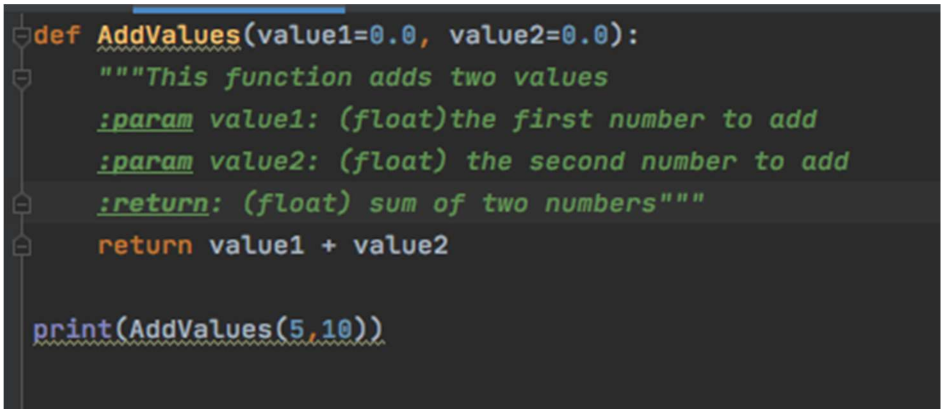
```
>>> x = makeOne()
>>> print x
1
>>> def addTen(myInt):
        myInt += 10
        return myInt

>>> x = 12
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__
othing', 'makeOne', 'x']
>>> y = addTen(x)
>>> print x,y
12 22
>>>
```

Figure 6. Global and local variables example

In the example above, you would think it would print out 22 twice, but when myInt is passed into the addTen function, the function actually makes a copy of that variable, as a local copy.

Function Documentation

As codes become more complicated, it helps to add headers, or descriptive function names, to help tell users what the functions purpose is. One thing you can do is add document strings to a function. These give programmers information on the parameters and return values of the function. By hovering the mouse over the variable, users can access this information. See **Figure 7** for an example of document strings.

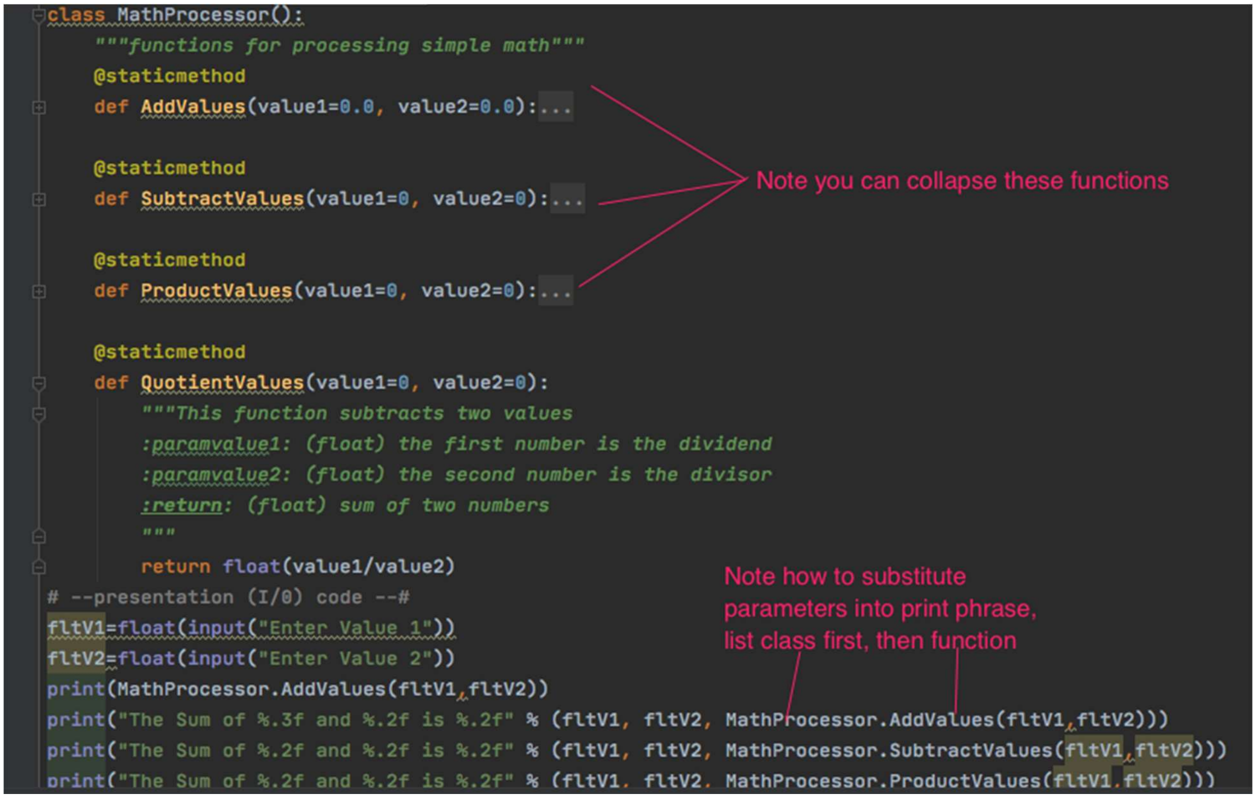


```python
def AddValues(value1=0.0, value2=0.0):
    """This function adds two values
    :param value1: (float)the first number to add
    :param value2: (float) the second number to add
    :return: (float) sum of two numbers"""
    return value1 + value2

print(AddValues(5,10))
```

Figure 7. Example of document string

Classes and Functions

Classes are a way of grouping functions, variables and constants. It is a good way to organize work, and can make your code easier to read and work with. In ***Figure 8***, you can see many functions under the MathProcessor() class.



```python
class MathProcessor():
    """functions for processing simple math"""
    @staticmethod
    def AddValues(value1=0.0, value2=0.0):...

    @staticmethod
    def SubtractValues(value1=0, value2=0):...

    @staticmethod
    def ProductValues(value1=0, value2=0):...

    @staticmethod
    def QuotientValues(value1=0, value2=0):
        """This function subtracts two values
        :paramvalue1: (float) the first number is the dividend
        :paramvalue2: (float) the second number is the divisor
        :return: (float) sum of two numbers
        """
        return float(value1/value2)
# --presentation (I/O) code --#
fltV1=float(input("Enter Value 1"))
fltV2=float(input("Enter Value 2"))
print(MathProcessor.AddValues(fltV1,fltV2))
print("The Sum of %.3f and %.2f is %.2f" % (fltV1, fltV2, MathProcessor.AddValues(fltV1,fltV2)))
print("The Sum of %.2f and %.2f is %.2f" % (fltV1, fltV2, MathProcessor.SubtractValues(fltV1,fltV2)))
print("The Sum of %.2f and %.2f is %.2f" % (fltV1, fltV2, MathProcessor.ProductValues(fltV1,fltV2)))
```

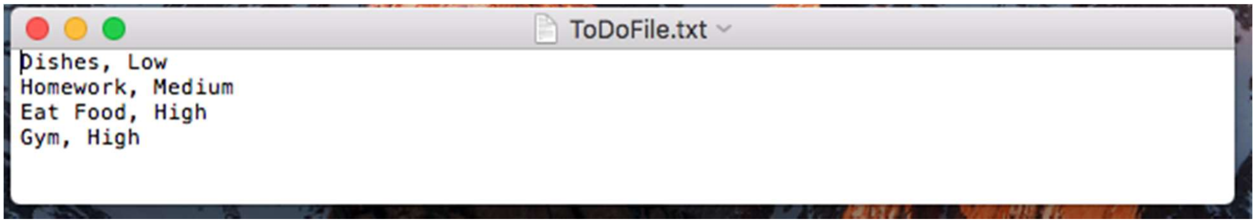
Note you can collapse these functions

Note how to substitute parameters into print phrase, list class first, then function

Figure 8. Example of using classes to organize functions.

Assignment

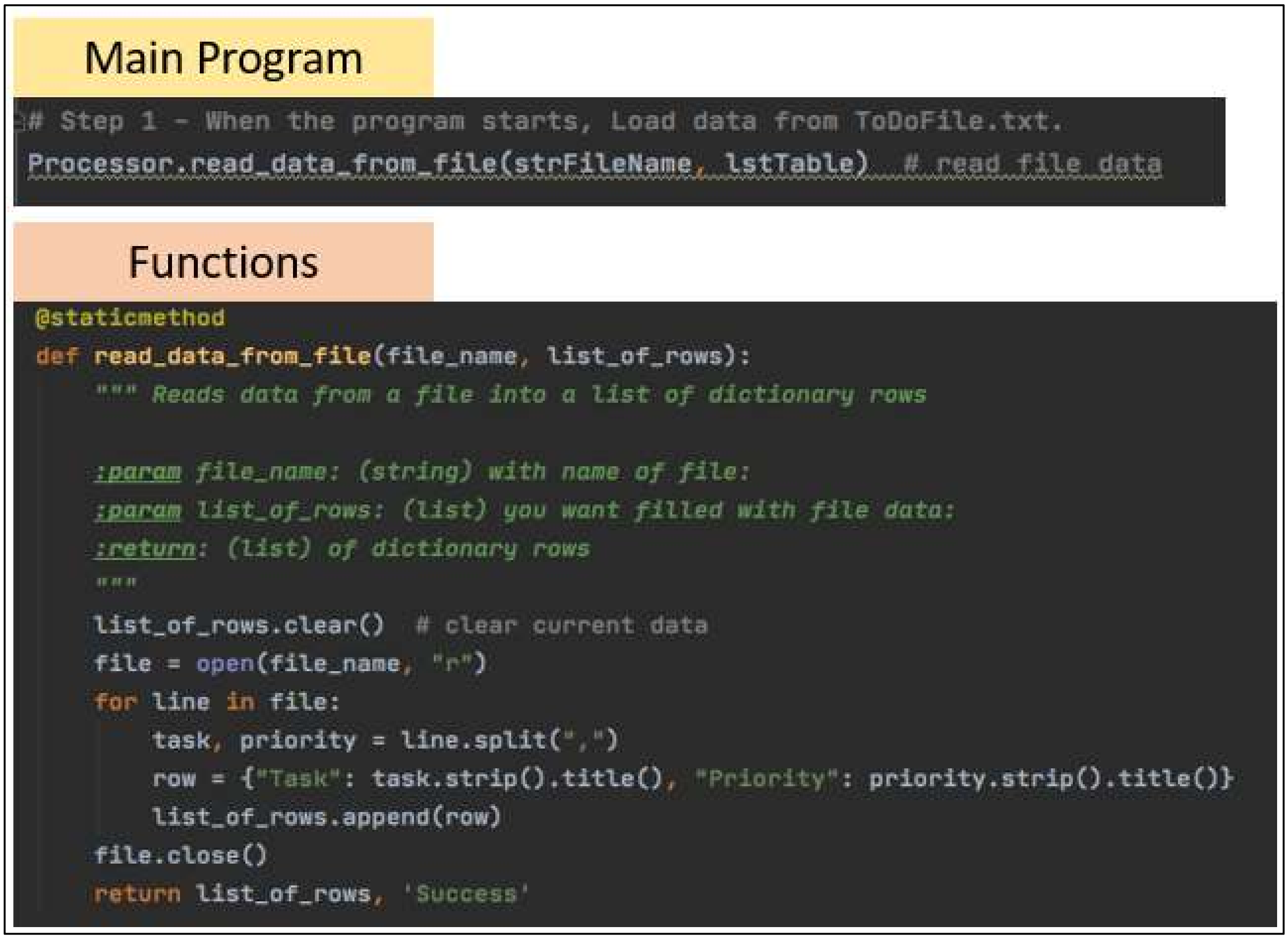
Similar to assignment 5, we are creating and modifying a task list with their corresponding priority. We begin the assignment with the .txt file already filled with a few task. See **Figure 9** to see the initial task list.



```
Dishes, Low
Homework, Medium
Eat Food, High
Gym, High
```

Figure 9. Initial task list that .txt file starts with

The first step in the program is reading the information into a list of dictionary rows.


Main Program

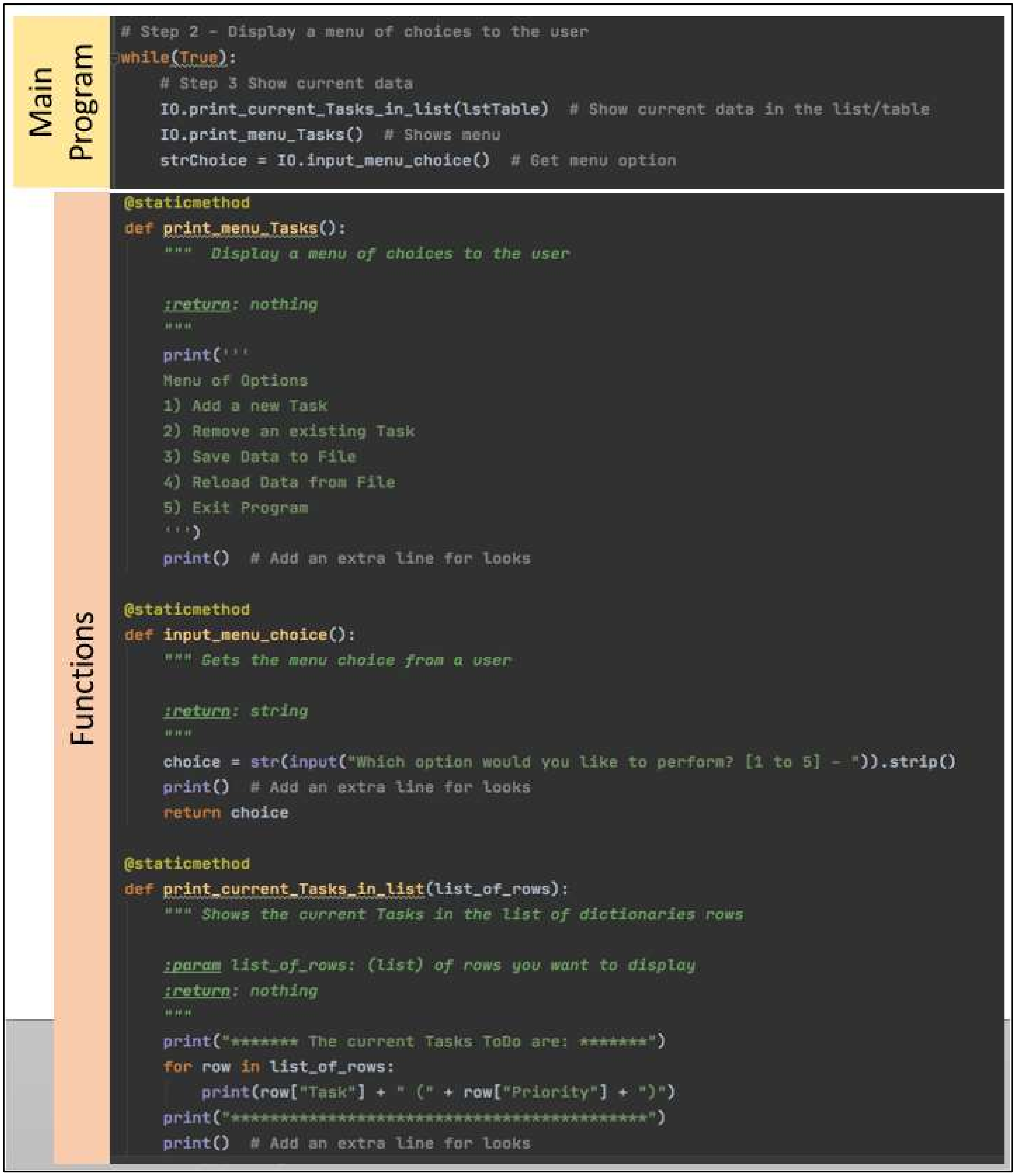
```
# Step 1 - When the program starts, Load data from ToDoFile.txt.
Processor.read_data_from_file(strFileName, lstTable)  # read file data
```

Functions

```
@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().title(), "Priority": priority.strip().title()}
        list_of_rows.append(row)
    file.close()
    return list_of_rows, 'Success'
```

Figure 10. Reading information from file.

After we read the data from the file, we have a list of dictionaries variable, lstTable. We then enter into out while loop where we display the current task list to the user, and then seek input from the user. See **Figure 11** for example of script.

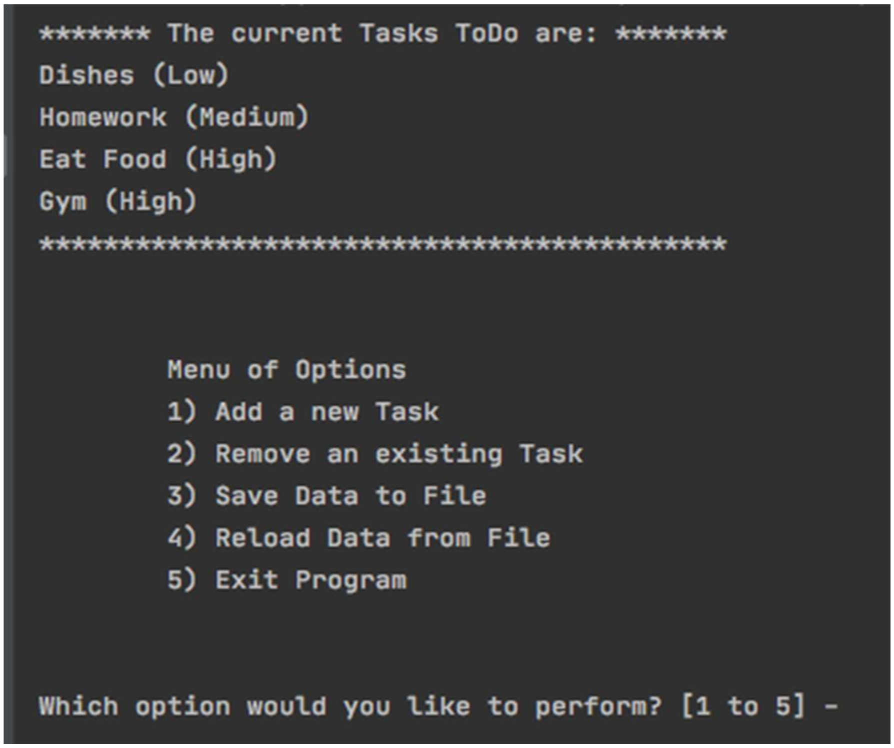


```
# Step 2 - Display a menu of choices to the user
while(True):
    # Step 3 Show current data
    IO.print_current_Tasks_in_list(lstTable)  # Show current data in the list/table
    IO.print_menu_Tasks()  # Shows menu
    strChoice = IO.input_menu_choice()  # Get menu option
```

```
@staticmethod
def print_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) Reload Data from File
    5) 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 5] - ")).strip()
    print()  # Add an extra line for looks
    return choice

@staticmethod
def print_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 11. Example of script.

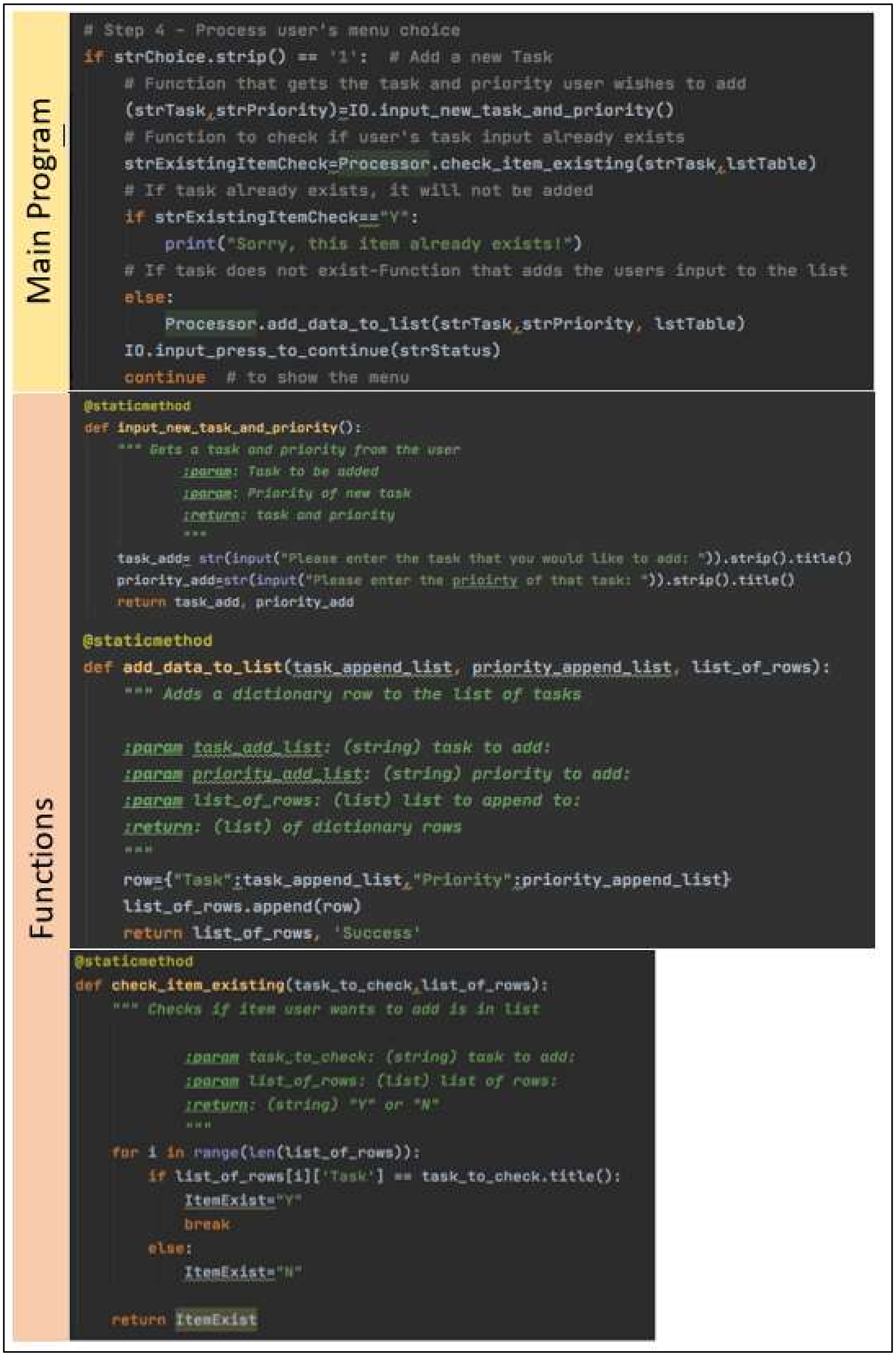
Below, in **Figure 12**, shows the script running in PyCharm.



```
******* The current Tasks ToDo are: *******
Dishes (Low)
Homework (Medium)
Eat Food (High)
Gym (High)
*******************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Reload Data from File
        5) Exit Program


Which option would you like to perform? [1 to 5] -
```

Figure 12. Example of script running in PyCharm before user chooses option.

Once the user enters their input for which menu item they select, that input is captured in the strChoice variable. If the user chooses option 1, three functions are then executed. In the first (input_new_task_and_priority), we collect the task and priority to add from the user. In the next (check_item_existing), we check to see if that is already existing in the list. If it is, the script tells the user that the item already exists. If not, the third function is executed, which adds the item to the list. See **Figure 13** for script.



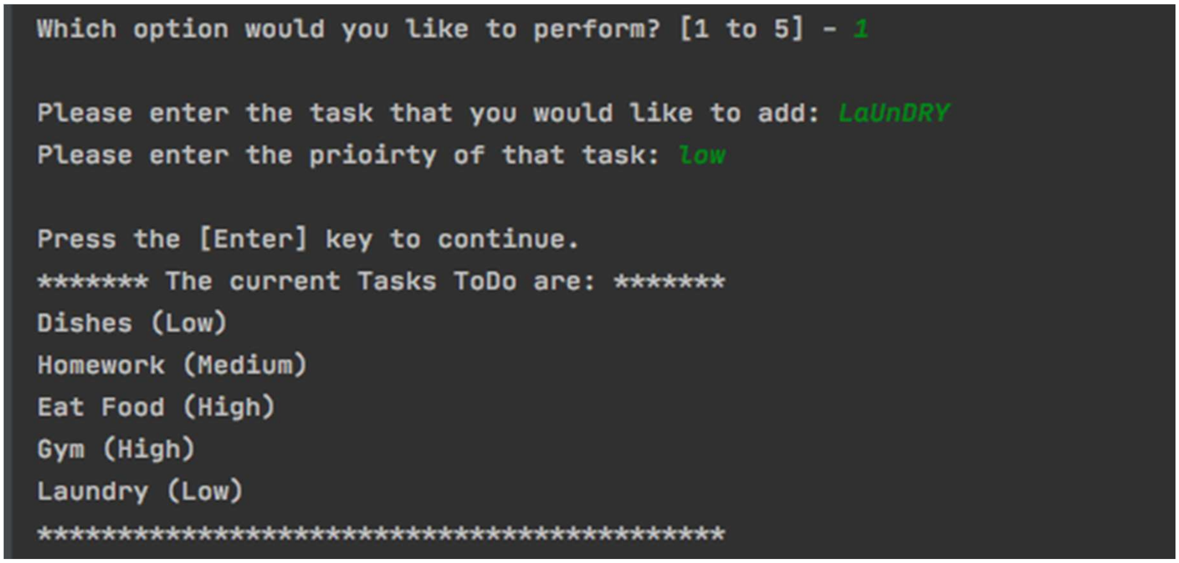
```
# Step 4 - Process user's menu choice
if strChoice.strip() == '1':  # Add a new Task
    # Function that gets the task and priority user wishes to add
    (strTask,strPriority)=IO.input_new_task_and_priority()
    # Function to check if user's task input already exists
    strExistingItemCheck=Processor.check_item_existing(strTask,lstTable)
    # If task already exists, it will not be added
    if strExistingItemCheck=="Y":
        print("Sorry, this item already exists!")
    # If task does not exist-Function that adds the users input to the list
    else:
        Processor.add_data_to_list(strTask,strPriority, lstTable)
    IO.input_press_to_continue(strStatus)
    continue  # to show the menu
```

```
@staticmethod
def input_new_task_and_priority():
    """ Gets a task and priority from the user
            :param: Task to be added
            :param: Priority of new task
            :return: task and priority
            """
    task_add= str(input("Please enter the task that you would like to add: ")).strip().title()
    priority_add=str(input("Please enter the prioirty of that task: ")).strip().title()
    return task_add, priority_add

@staticmethod
def add_data_to_list(task_append_list, priority_append_list, list_of_rows):
    """ Adds a dictionary row to the list of tasks

    :param task_add_list: (string) task to add:
    :param priority_add_list: (string) priority to add:
    :param list_of_rows: (list) list to append to:
    :return: (list) of dictionary rows
    """
    row={"Task":task_append_list,"Priority":priority_append_list}
    list_of_rows.append(row)
    return list_of_rows, 'Success'
```

```
@staticmethod
def check_item_existing(task_to_check,list_of_rows):
    """ Checks if item user wants to add is in list

            :param task_to_check: (string) task to add:
            :param list_of_rows: (list) list of rows:
            :return: (string) "Y" or "N"
            """
    for i in range(len(list_of_rows)):
        if list_of_rows[i]['Task'] == task_to_check.title():
            ItemExist="Y"
            break
        else:
            ItemExist="N"

    return ItemExist
```

Figure 13. Main script and functions called if user chooses option 1.

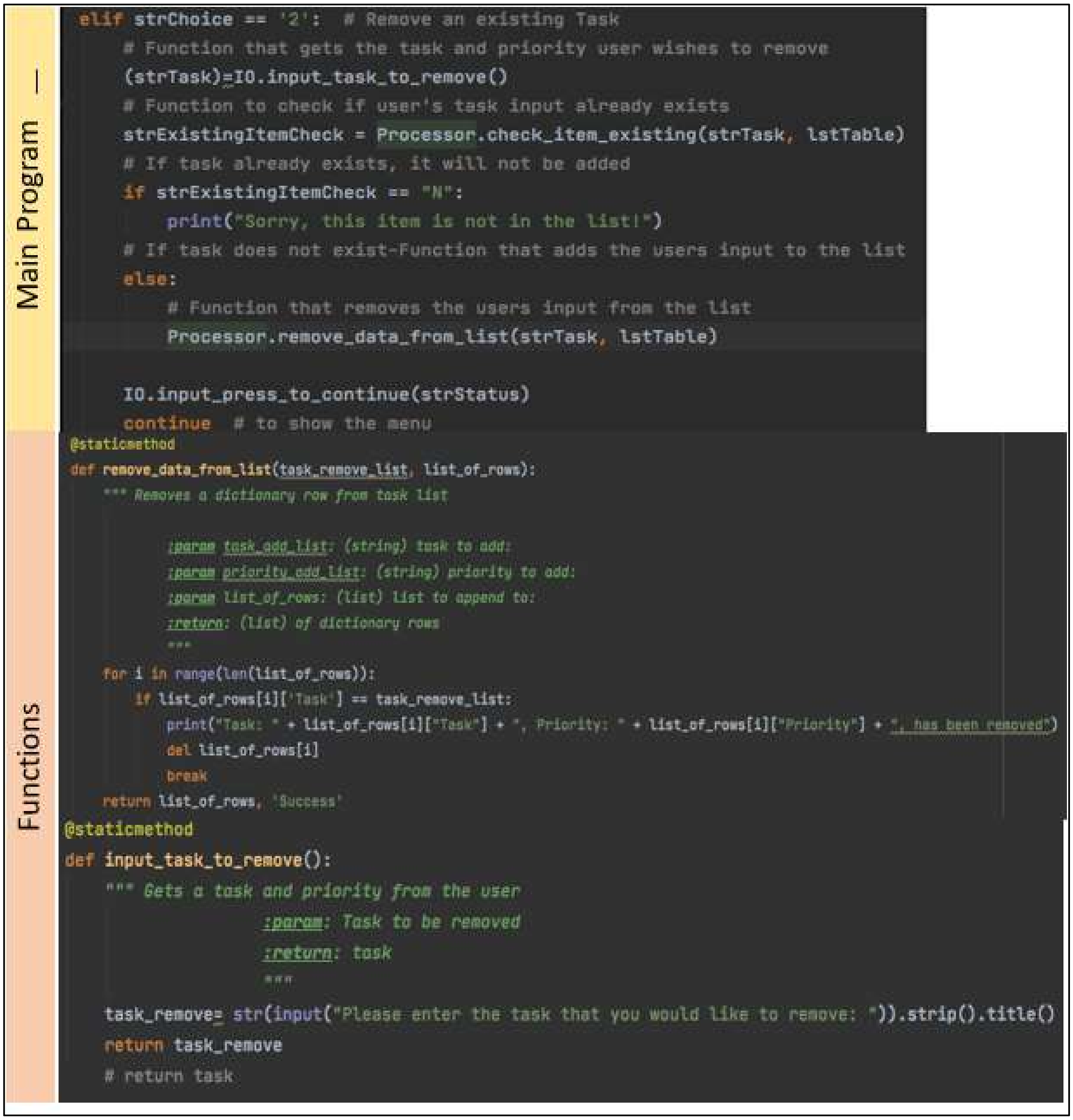
In **Figure 14**, you can see the task and its respective priority I am adding when running the scripts in PyCharm. Note that the script automatically changes the case of task and priority to title for using the .title() function.



```
Which option would you like to perform? [1 to 5] - 1

Please enter the task that you would like to add: LaUnDRY
Please enter the prioirty of that task: low

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Dishes (Low)
Homework (Medium)
Eat Food (High)
Gym (High)
Laundry (Low)
*******************************************
```

Figure 14. Script running in PyCharm when option 1 is chosen.

If the user chooses option 2, which is to remove a task from the list, the following is executed (**Figure 15**). Note the reuse of the check_item_existing function.

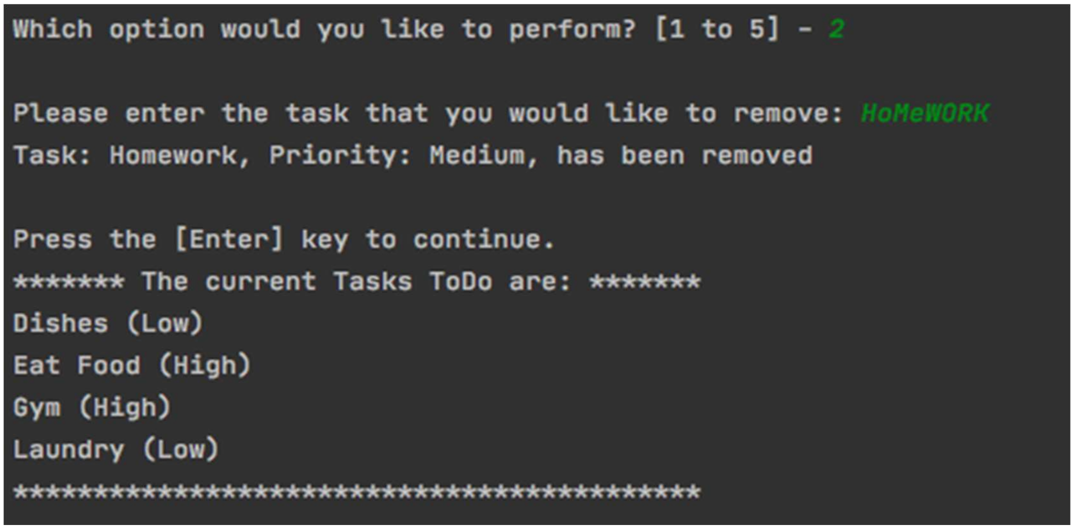


```
elif strChoice == '2':  # Remove an existing Task
    # Function that gets the task and priority user wishes to remove
    (strTask)=IO.input_task_to_remove()
    # Function to check if user's task input already exists
    strExistingItemCheck = Processor.check_item_existing(strTask, lstTable)
    # If task already exists, it will not be added
    if strExistingItemCheck == "N":
        print("Sorry, this item is not in the list!")
    # If task does not exist-Function that adds the users input to the list
    else:
        # Function that removes the users input from the list
        Processor.remove_data_from_list(strTask, lstTable)

    IO.input_press_to_continue(strStatus)
    continue  # to show the menu
```

```
@staticmethod
def remove_data_from_list(task_remove_list, list_of_rows):
    """ Removes a dictionary row from task list

            :param task_add_list: (string) task to add:
            :param priority_add_list: (string) priority to add:
            :param list_of_rows: (list) list to append to:
            :return: (list) of dictionary rows
            """
    for i in range(len(list_of_rows)):
        if list_of_rows[i]['Task'] == task_remove_list:
            print("Task: " + list_of_rows[i]["Task"] + ", Priority: " + list_of_rows[i]["Priority"] + ", has been removed")
            del list_of_rows[i]
            break
    return list_of_rows, 'Success'
@staticmethod
def input_task_to_remove():
    """ Gets a task and priority from the user
                :param: Task to be removed
                :return: task
                """
    task_remove= str(input("Please enter the task that you would like to remove: ")).strip().title()
    return task_remove
    # return task
```

Figure 15. Functions and main script that drive option 2.

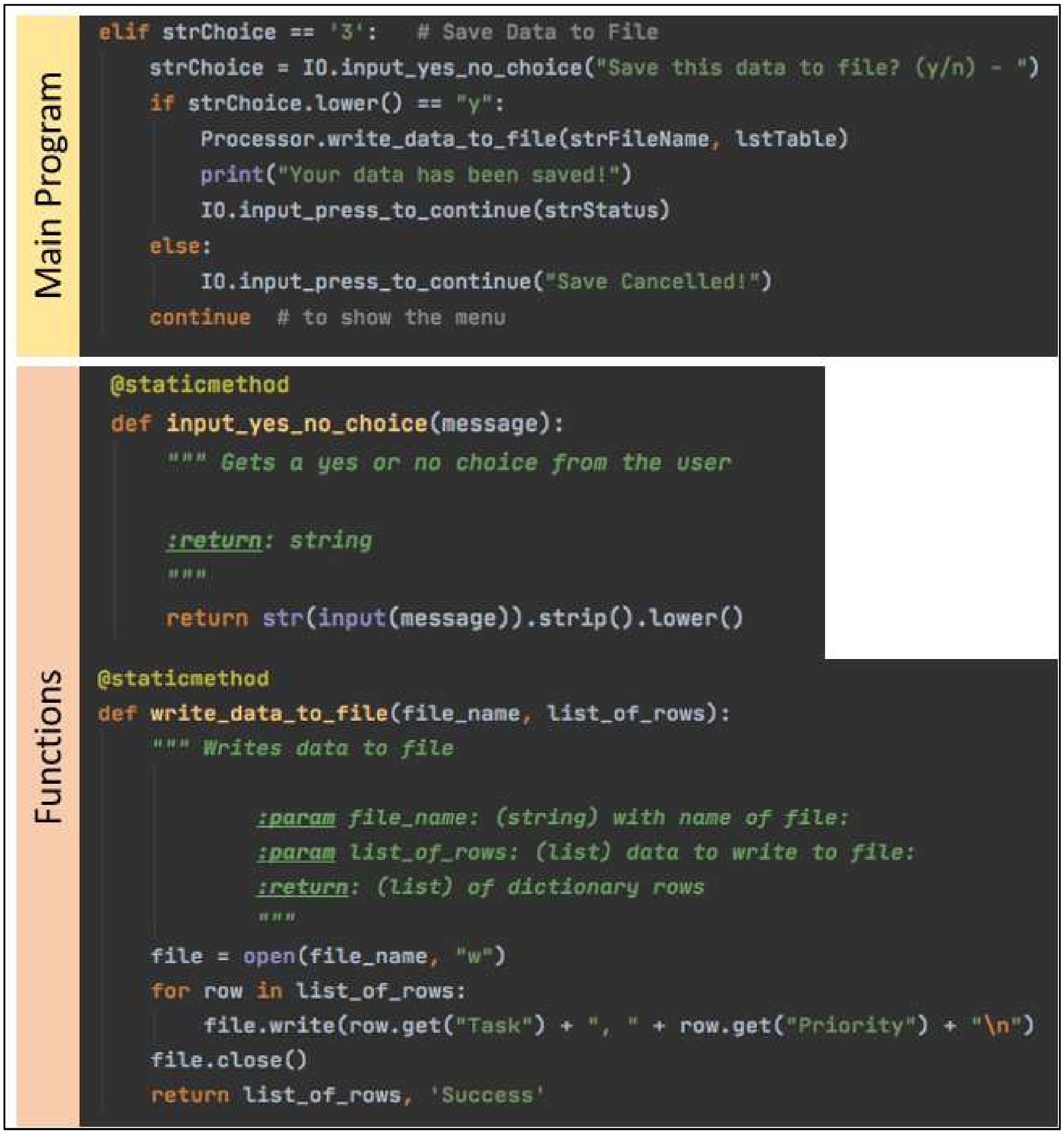
In Figure 16 below, you see the task entered was remove, and the subsequent modified list reprinted.



```
Which option would you like to perform? [1 to 5] - 2

Please enter the task that you would like to remove: HoMeWORK
Task: Homework, Priority: Medium, has been removed

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Dishes (Low)
Eat Food (High)
Gym (High)
Laundry (Low)
*******************************************
```

Figure 16. Option 3 selection running in PyCharm

If the user choses option 3, which is to save the file, the following script is executed (**Figure 17**).



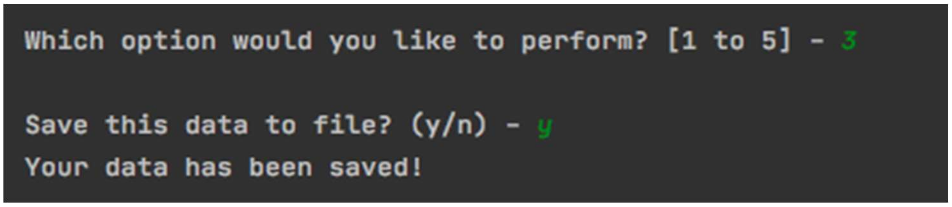
```
elif strChoice == '3':   # Save Data to File
    strChoice = IO.input_yes_no_choice("Save this data to file? (y/n) - ")
    if strChoice.lower() == "y":
        Processor.write_data_to_file(strFileName, lstTable)
        print("Your data has been saved!")
        IO.input_press_to_continue(strStatus)
    else:
        IO.input_press_to_continue("Save Cancelled!")
    continue  # to show the menu
```

```
@staticmethod
def input_yes_no_choice(message):
    """ Gets a yes or no choice from the user

    :return: string
    """
    return str(input(message)).strip().lower()
```

```
@staticmethod
def write_data_to_file(file_name, list_of_rows):
    """ Writes data to file

            :param file_name: (string) with name of file:
            :param list_of_rows: (list) data to write to file:
            :return: (list) of dictionary rows
            """
    file = open(file_name, "w")
    for row in list_of_rows:
        file.write(row.get("Task") + ", " + row.get("Priority") + "\n")
    file.close()
    return list_of_rows, 'Success'
```

Figure 17. Functions and main scripts running if option 3 is selected.

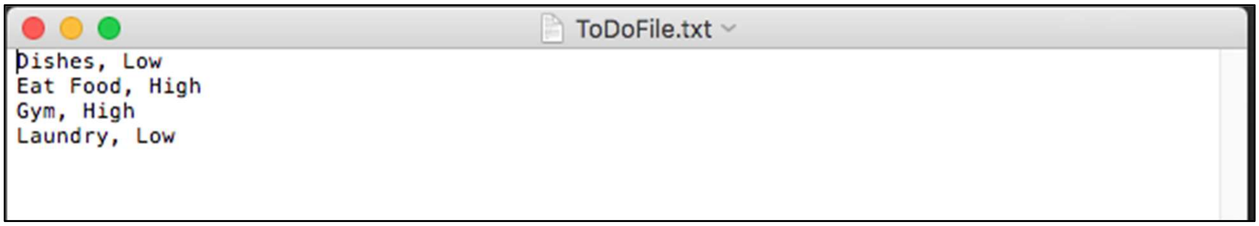
In **Figure 18** below, you can see this script executing in PyCharm.



```
Which option would you like to perform? [1 to 5] - 3

Save this data to file? (y/n) - y
Your data has been saved!
```

Figure 18. Option 3 selection in PyCharm.

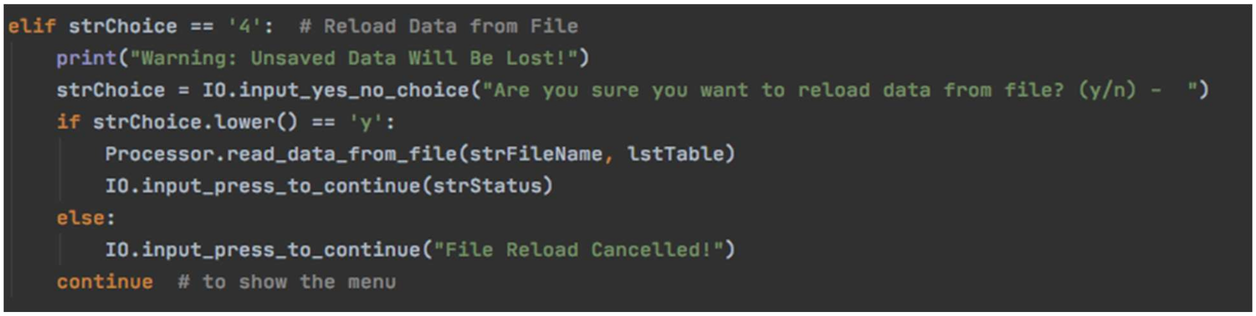
The modified list has now been saved into the .txt file, shown in **Figure 19**.



```
Dishes, Low
Eat Food, High
Gym, High
Laundry, Low
```

Figure 19. Saved list in .txt file.

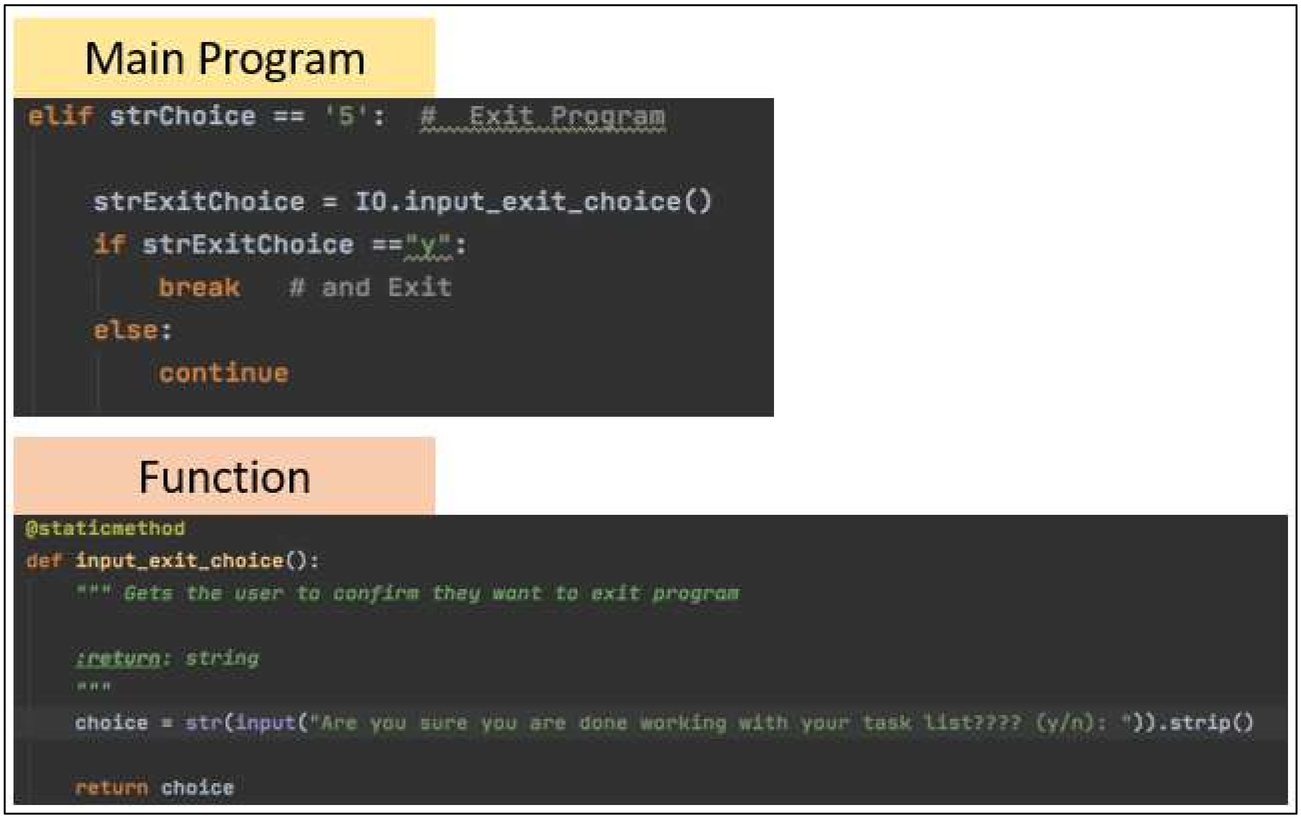
If the user chooses option 4, the data will reload from the file. See script shown in **Figure 20**. Note that with this option, we warn the user that any changes made to the list will not be saved. We then execute the functions previously shown in **Figure 10**.



```
elif strChoice == '4':  # Reload Data from File
    print("Warning: Unsaved Data Will Be Lost!")
    strChoice = IO.input_yes_no_choice("Are you sure you want to reload data from file? (y/n) -  ")
    if strChoice.lower() == 'y':
        Processor.read_data_from_file(strFileName, lstTable)
        IO.input_press_to_continue(strStatus)
    else:
        IO.input_press_to_continue("File Reload Cancelled!")
    continue  # to show the menu
```

Figure 21. Script of option 4 selection.

If option 5 is selected, we ask the user one last time if they are sure that that is what they wish to do. I have created a new function in the IO class that does this. See **Figure 22** below.



Main Program

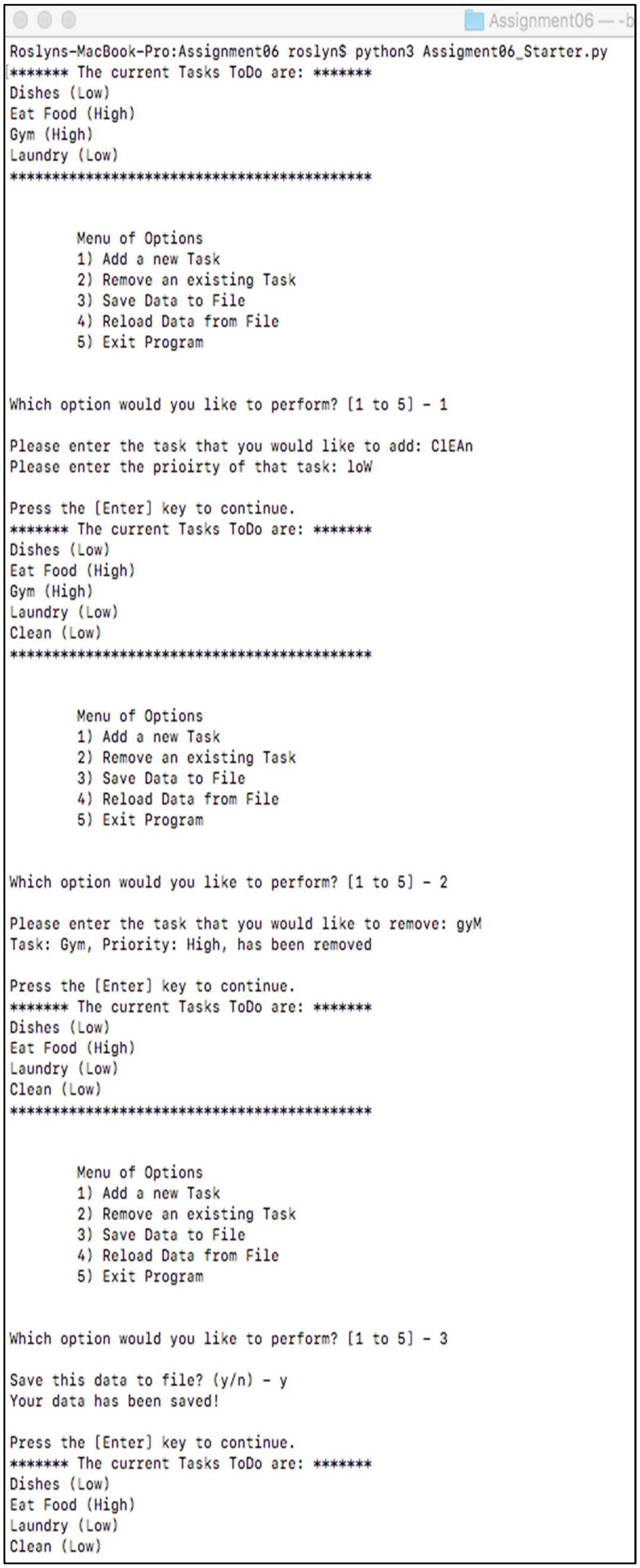
```
elif strChoice == '5':  #  Exit Program

    strExitChoice = IO.input_exit_choice()
    if strExitChoice =="y":
        break    # and Exit
    else:
        continue
```

Function

```
@staticmethod
def input_exit_choice():
    """ Gets the user to confirm they want to exit program

    :return: string
    """
    choice = str(input("Are you sure you are done working with your task list???? (y/n): ")).strip()

    return choice
```

Figure 22. Addition function to confirm that user wants to exit

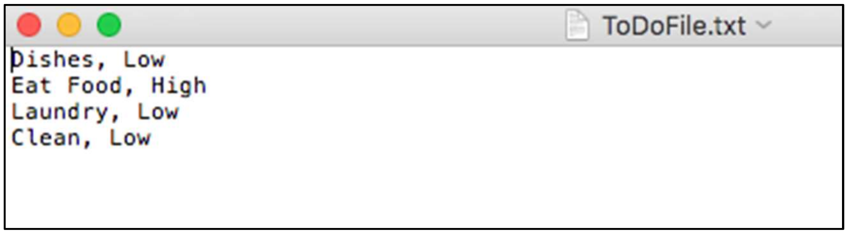
Next I tested the script to make sure that it would successfully run in my mac terminal. See **Figure 23**.



```
Roslyns-MacBook-Pro:Assignment06 roslyn$ python3 Assigment06_Starter.py
******* The current Tasks ToDo are: *******
Dishes (Low)
Eat Food (High)
Gym (High)
Laundry (Low)
*******************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Reload Data from File
        5) Exit Program


Which option would you like to perform? [1 to 5] - 1

Please enter the task that you would like to add: ClEAn
Please enter the prioirty of that task: loW

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Dishes (Low)
Eat Food (High)
Gym (High)
Laundry (Low)
Clean (Low)
*******************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Reload Data from File
        5) Exit Program


Which option would you like to perform? [1 to 5] - 2

Please enter the task that you would like to remove: gyM
Task: Gym, Priority: High, has been removed

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Dishes (Low)
Eat Food (High)
Laundry (Low)
Clean (Low)
*******************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Reload Data from File
        5) Exit Program


Which option would you like to perform? [1 to 5] - 3

Save this data to file? (y/n) - y
Your data has been saved!

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Dishes (Low)
Eat Food (High)
Laundry (Low)
Clean (Low)
```

Figure 23. Script running from Mac terminal.

You can see that this revised list has now been saved to the file (Figure 24).



```
Dishes, Low
Eat Food, High
Laundry, Low
Clean, Low
```

Figure 24. Modified list after running script from terminal.

Summary

In this module, we learned how to use functions and how they can be really helpful to organize and troubleshoot code. In this module, we also began to organize our functions into classes, which help organize our code even further and help to make our code easier to pick up for users down the road. We learned the importance of differentiating between global and local variables. We also learned how to create a GitHub webpage.