Barbara Spadavecchia
February 22, 2022
IT FTN 110 A
Assignment06
https://github.com/Barb4000/IntroToProg-Python-Mod06

Creating Scripts Using Functions and Classes

Introduction

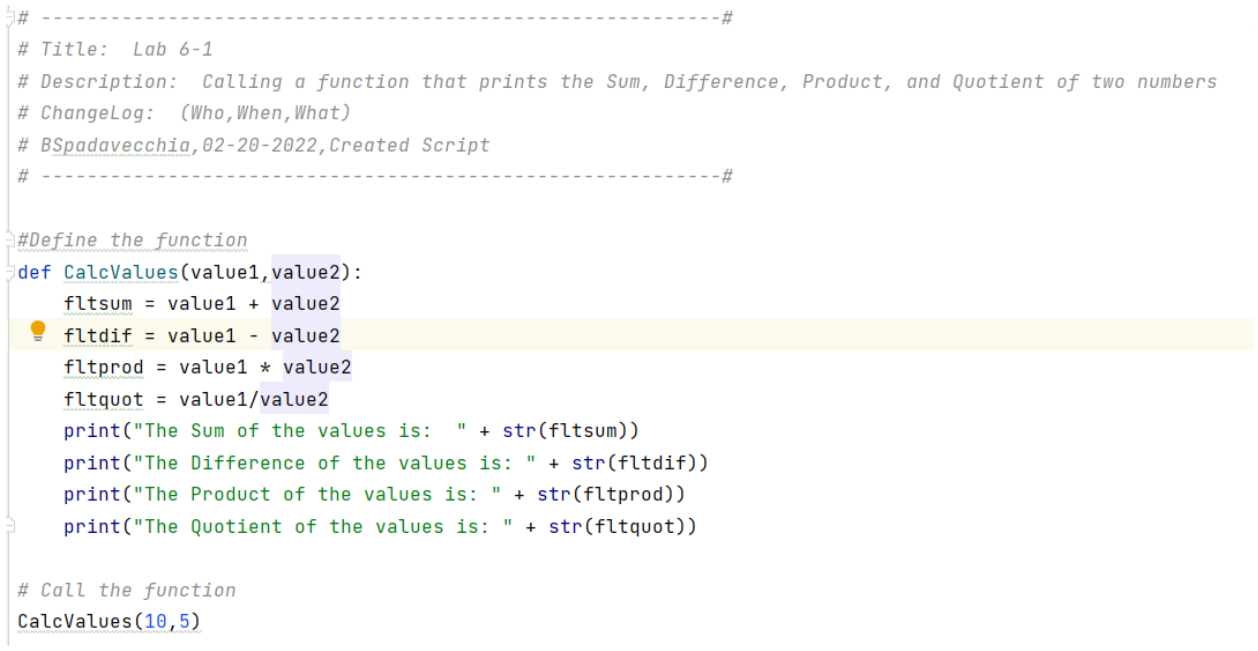
This paper discusses how to use functions to group one or more statements. Functions are useful to break up large programs into smaller manageable sections of code. Programs that use functions can be easier to understand and maintain and follow the best practice of "Separation of Concerns." The use of functions was practiced on a "ToDo" list program from the previous module. This assignment also includes a brief description of adding a GitHub Web-Page.

Functions

Functions are a way of grouping one or more statements. They must be defined before they can be used. A function is called by naming the function followed by (). Functions can have parameters that allow values to pass into the function for processing. If values are passed into parameter they are called "arguments". Many arguments can be passed to a function. In Lab 6-1 a function was created that printed the sum, difference, product, and quotient of two numbers. The code worked by defining the function "def CalcValues" to pass two values into the function where they were processed when the function was called. Return values are used to pass back data to the part of the program that called their function. Functions can return one or more values. Return values make a function act as an expression where results can be used immediately without being placed in a variable. Functions allow for the separation of processing of values and presentation. There is no presentation code when using the return option. A function document header or "docstring" provides additional notes.

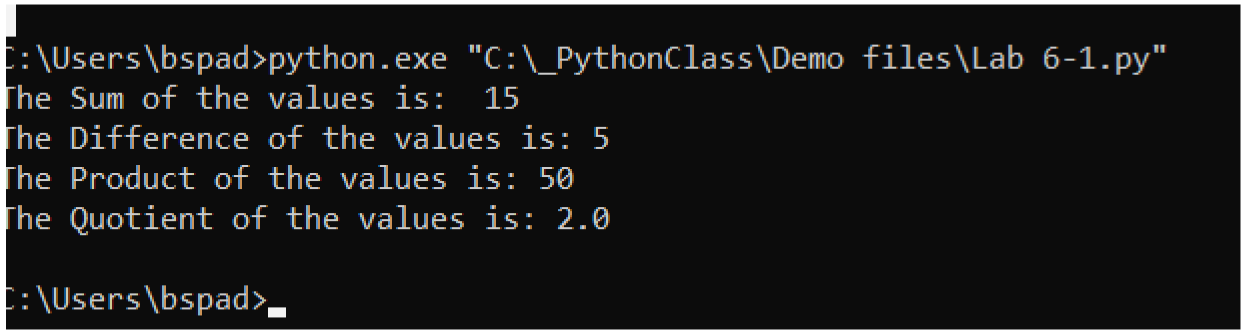
Parameters and Return Values

Parameters are variable names inside the header of a function. The parameters contain the values sent to a function from the function's call through its arguments. The keyword "None" indicates the absence of a parameter value and is often used as a default argument. The return value can pass back one or more values to the part that called it. The function will end after the return statement. The return statement allows the function to act as an expression. There is no presentation code in a function using return option and this helps with the separation of concerns by separating processing from presentation.



```
# -----------------------------------------------------------#
# Title:  Lab 6-1
# Description:  Calling a function that prints the Sum, Difference, Product, and Quotient of two numbers
# ChangeLog:  (Who,When,What)
# BSpadavecchia,02-20-2022,Created Script
# -----------------------------------------------------------#

#Define the function
def CalcValues(value1,value2):
    fltsum = value1 + value2
    fltdif = value1 - value2
    fltprod = value1 * value2
    fltquot = value1/value2
    print("The Sum of the values is:  " + str(fltsum))
    print("The Difference of the values is: " + str(fltdif))
    print("The Product of the values is: " + str(fltprod))
    print("The Quotient of the values is: " + str(fltquot))

# Call the function
CalcValues(10,5)
```

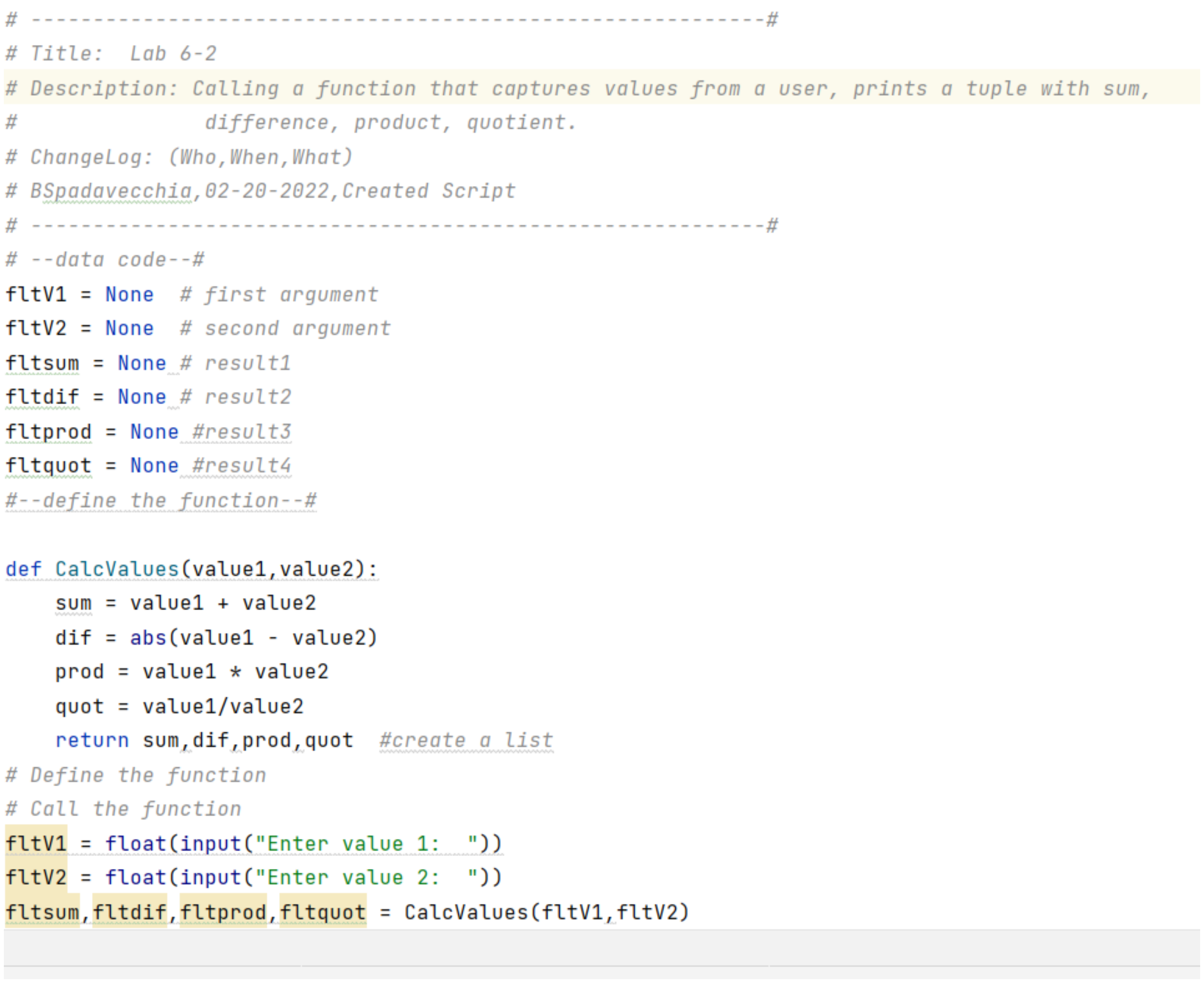
Figure 1 Lab 6-1



```
:\Users\bspad>python.exe "C:\_PythonClass\Demo files\Lab 6-1.py"
he Sum of the values is:  15
he Difference of the values is: 5
he Product of the values is: 50
he Quotient of the values is: 2.0

:\Users\bspad>
```

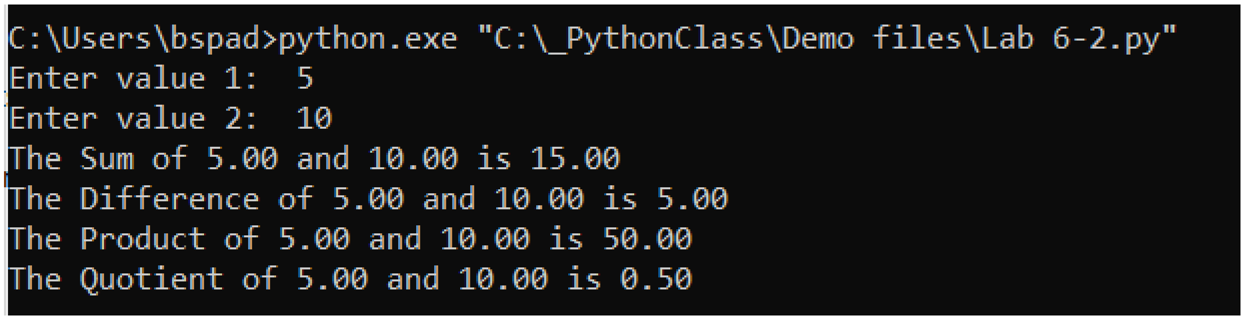
Figure 2 Lab 6-1 Running from the Command Prompt

Lab 6-2 captures two values from a user which then return a tuple of results including: sum, difference, product, and quotient.



```python
# ----------------------------------------------------------#
# Title:  Lab 6-2
# Description: Calling a function that captures values from a user, prints a tuple with sum,
#                difference, product, quotient.
# ChangeLog: (Who,When,What)
# BSpadavecchia,02-20-2022,Created Script
# ----------------------------------------------------------#
# --data code--#
fltV1 = None  # first argument
fltV2 = None  # second argument
fltsum = None # result1
fltdif = None # result2
fltprod = None #result3
fltquot = None #result4
#--define the function--#

def CalcValues(value1,value2):
    sum = value1 + value2
    dif = abs(value1 - value2)
    prod = value1 * value2
    quot = value1/value2
    return sum,dif,prod,quot  #create a list
# Define the function
# Call the function
fltV1 = float(input("Enter value 1:  "))
fltV2 = float(input("Enter value 2:  "))
fltsum,fltdif,fltprod,fltquot = CalcValues(fltV1,fltV2)
```

Figure 3 Lab 6-2 Script for Returning Tuples



```
C:\Users\bspad>python.exe "C:\_PythonClass\Demo files\Lab 6-2.py"
Enter value 1:  5
Enter value 2:  10
The Sum of 5.00 and 10.00 is 15.00
The Difference of 5.00 and 10.00 is 5.00
The Product of 5.00 and 10.00 is 50.00
The Quotient of 5.00 and 10.00 is 0.50
```

Figure 4 Lab 6-2 Running from the Command Prompt

Working with Arguments

Arguments make a function perform different actions or return different results. Positional arguments follow the order and if they are switched around errors can happen. Named arguments are easier to read and give fewer errors. Default parameter values can be set. They will be used if there is no argument for a parameter.

Global vs. Local Variables

Variables that are declared in a function are considered local to the function containing them. They cannot be accessed outside of the function. Variables declared in a "body" of script are global and can be used anywhere in the script. If a global variable is used inside of a function, it should start with a "g" to identify it as global. It is important to use the keyword "global" when you assign a value to a variable of the same name. Local variables are commonly named without a prefix since they are used within the function. This helps prevent shadowing.

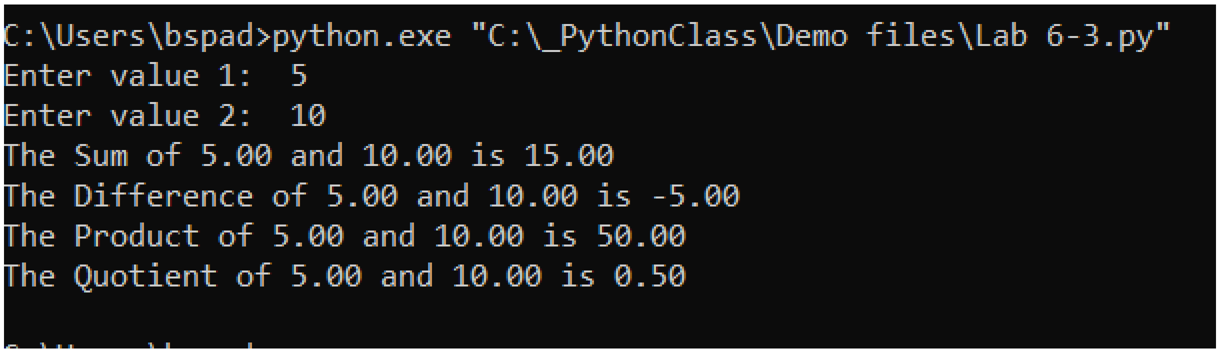
Classes and Functions

Classes are a way to group functions, variables, and constants. Lab 6-3 used the class "MathProcessor" for all the functions related to processing simple math. This lab used functions to add and subtract, multiply, and divide. The values were returned from the functions to the presentation I/O section. The separation of the processing code by function helps organize the code. Each process is broken down into all its individual functions. This enables code to be re-used and easily updated. The separation of concerns are followed when the functions for processing are separated from presentation. Lab 6-3 gives examples of using "docstrings" to describe the functions and states what the values, how they are processed, and what values are returned.

```
#------------------------------------------------------------#
# Title:  Lab 6-3
# Description:  Creating 4 Classes of Functions returning Sum, Difference,
Product, Quotient
# ChangeLog:  (Who,When,What)
# BSpadavecchia,02-20-2022,Created Script
# ------------------------------------------------------------#
#--data code--#
fltV1 = 0.0 #First number for calculation
fltV2 = 0.0 #Second number for calculation
#--processing code--#


class MathProcessor():
    """functions for processing simple math"""

    @staticmethod
    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 float(value1 + value2)
    @staticmethod
    def SubtractValues(value1=0, value2=0):
        """This function subtracts two values

                :param value1(float) the first number to subtract
                :param value2(float) the second number to subtract
                :return: (float) difference of two numbers
```

```python
                """
        return float(value1 - value2)

    @staticmethod
    def MultiplyValues(value1=0, value2=0):
        """This function multiples two values

                :param value1(int or float) the first number to multiply
                :param value2(int or float) the second number to multiply
                :return: (int or float) product of two numbers
                """
        return float(value1 * value2)

    @staticmethod
    def DivideValues(value1=0, value2=0):
        """This function divides two values

                :param value1(float) the first number to divide
                :param value2(float) the second number to divide
                :return: (float) quotient of two numbers
                """
        return float(value1 / value2)
#--presentation (I/O code--#
fltV1 = float(input("Enter value 1:  "))
fltV2 = float(input("Enter value 2:  "))
print("The Sum of %.2f and %.2f is %.2f" %
(fltV1,fltV2,MathProcessor.AddValues(fltV1,fltV2)))
print("The Difference of %.2f and %.2f is %.2f" %
(fltV1,fltV2,MathProcessor.SubtractValues(fltV1,fltV2)))
print("The Product of %.2f and %.2f is %.2f" % (fltV1,
fltV2,MathProcessor.MultiplyValues(fltV1,fltV2)))
print("The Quotient of %.2f and %.2f is %.2f" % (fltV1,
fltV2,MathProcessor.DivideValues(fltV1,fltV2)))
```

Figure 5 Lab 6-3 Script



```
C:\Users\bspad>python.exe "C:\_PythonClass\Demo files\Lab 6-3.py"
Enter value 1:   5
Enter value 2:   10
The Sum of 5.00 and 10.00 is 15.00
The Difference of 5.00 and 10.00 is -5.00
The Product of 5.00 and 10.00 is 50.00
The Quotient of 5.00 and 10.00 is 0.50
```

Figure 6 Lab 6-3 Running from the Command Prompt

https://www.learnpython.org/en/Functions

This website described a function as a way to divide code into blocks. Functions help organize code and allow for other programmers to interface with the code. Keywords used with functions are: for, if, while. Functions in python are defined with the keyword “def” followed by the function’s name.

Variables can be passed to the function from the caller. Functions can return a value to the caller by using the keyword "return". This website gave short examples of functions that could also be executed. There was also an exercise to try out that included a solution.

https://www.youtube.com/watch?v=qO4ZN5uZSVg

This youtube video introduces functions. It clearly goes over how variables that are stored within a function will be deleted when the function completes.

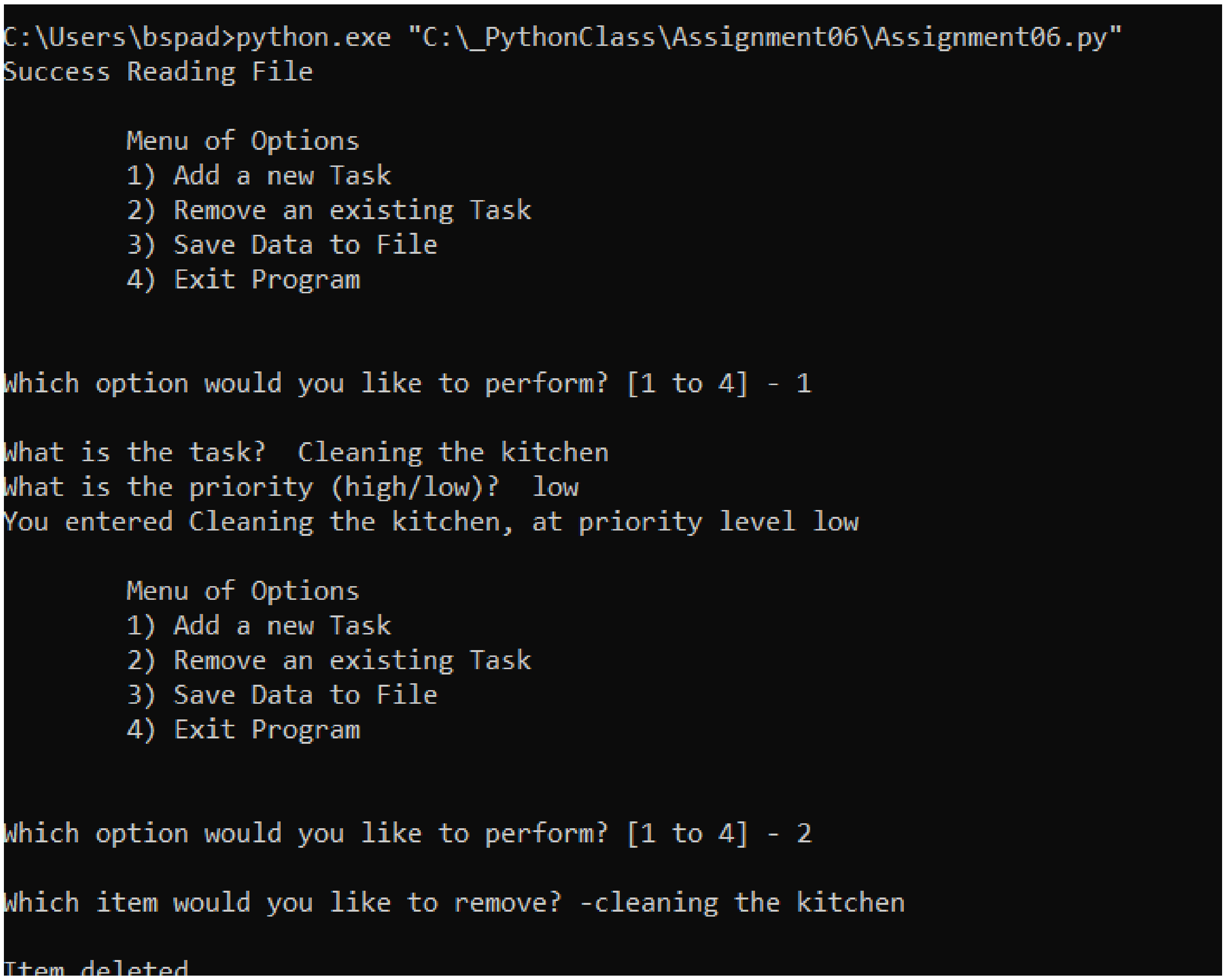
This script uses functions to create a "ToDo" List. The script is divided into sections: Data, Processioning, and Presentation. Functions are defined at the start of the Processioning section. The functions are organized by class. The "class Processor" functions perform the processing tasks of reading the data from the file, adding data to the list, removing data from the list, and saving data to a file. The "class I/O" performs input and output tasks. The portion has a function to display the menu, another function to get a selection choice from the user for the menu. Each selection choice then has its own function. The main body of the script is the part that calls the functions. Each of the functions is read into memory before they are processed. Return values pass data to the main body part of the script as the program runs.

Using the PyCharm Debugger

The "Debug mode" in PyCharm is found next to the "Run" option. The Debug mode was helpful in checking out how the functions were running. I set a "breakpoint" to check what was happening in the code. It was helpful to find missing syntax without having to go through every line. The options to use the "Step into My Code" button allows you to walk through the lines that perform actions and skip the comments. It also was useful to see how the variables were being typed by python and understand why code would not run if it was typed as a string.

```
# ---------------------------------------------------------------------------
- #
# Title: Assignment 06
# Description: Working with functions in a class,
#              When the program starts, load each "row" of data
#              in "ToDoList.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
# BSpadavecchia,02.21.2022,Modified code to complete assignment 06
# ---------------------------------------------------------------------------
- #

# Data ----------------------------------------------------------------------
#
# Declare variables and constants
file_name_str = "ToDoList.txt"  # The name of the data file
file_obj = None  # An object that represents a file
row_dic = {}  # A row of data separated into elements of a dictionary
{Task,Priority}
table_lst = []  # A list that acts as a 'table' of rows
choice_str = ""  # Captures the user option selection
# strTask = ""  # Captures user task data
# strPriority = ""  # Captures user priority for task
# strStat = ""  # Captures user processing status


# Processing  --------------------------------------------------------------
#
class Processor:
    """  Performs Processing tasks """

    @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()
        print("Success Reading File")
        return list_of_rows

    @staticmethod
    def add_data_to_list(task, priority, list_of_rows):
        row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
        list_of_rows.append(row)
        return list_of_rows

    @staticmethod
    def remove_data_from_list(task, list_of_rows):
        """ Removes data from a list of dictionary rows"""
        for row in list_of_rows:
            if row["Task"].lower() == task.lower():
                list_of_rows.remove(row)
                print("Item deleted")
        return list_of_rows

    @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")
        file.close()
        return list_of_rows

# Presentation (Input/Output)  -------------------------------------------- #

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

    @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

    @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

   # @staticmethod
   #    """User choice yes or no
    #    :return:  string
    #    """
    #    return str(input(message)).strip().lower()
   # @staticmethod
   # def input_press_to_continue(optional_message=''):
     #   """Pause program and show a message before continuing
     #   :param optional_message:  An optional message you want to display
      #  :return: nothing
     #   """
      #  print(optional_message)
     #   input('Press the [Enter] key to continue')

    @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
        """

        task = str(input("What is the task?  ")).strip()
        priority = str(input("What is the priority (high/low)?  ")).strip()
        print("You entered " + task + ", at priority level " + priority)
        return task, priority

    @staticmethod
    def input_task_to_remove():
        """  Gets the task name to be removed from the list
        :return: (string) with task
        """
        task = str(input("Which item would you like to remove? -")).strip()
        print()
        return task

# Main Body of Script  ------------------------------------------------------
#

# 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


# Step 2 - Display a menu of choices to the user
while (True):
    # Step 3 Show current data
    IO.output_current_tasks_in_listlist_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

    # Step 4 - Process user's menu choice
    if choice_str == '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

    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

    elif choice_str == '3':  # Save Data to File
        #choice_str = IO.input_yes_no_choice("Save this data to file:  (y/n)
- ")
        #if choice_str.lower() == "y":
        table_lst = Processor.write_data_to_file(file_name=file_name_str,
list_of_rows=table_lst)
        print("Data Saved!")
        #else:
            # IO.input_press_to_continue("Save Cancelled")
        continue  # to show the menu

    elif choice_str == '4':  # Exit Program
        print("Goodbye!")
        break  # by exiting loop
```

Figure 7 Script for the "ToDo List" program



```
C:\Users\bspad>python.exe "C:\_PythonClass\Assignment06\Assignment06.py"
Success Reading File

        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

What is the task?  Cleaning the kitchen
What is the priority (high/low)?  low
You entered Cleaning the kitchen, at priority level 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] - 2

Which item would you like to remove? -cleaning the kitchen

Item deleted
```

Figure 8 "ToDoList" program running on the command prompt

Creating a GitHub Webpage

The process involved making a new GitHub repository along with a new folder. The website was configured by using the “Setting” tab. The “main branch” option in the “source” dropdown box started the website generation. This used to be called “master branch/docs folder,” but this has been updated by GitHub. The website can be modified by editing the page later.

Conclusion

The process of working with functions was very beneficial to help see ways to organize code. I can see the advantages in having separate functions in dividing up the workload as well as an effective way to simplify debugging. It was very clear to see how the code could be traced from the main where the function was called. The process would have been simpler to consider how each process could be divided up into sections before writing actual code. It was awkward to go back and try to further separate out more of the presentation functions when there was existing code.