

Name: Natta Panapornsirikun

Date: 05/20/2023

Course: IT FDN 110 A: Foundation Of Programming Python

Assignment 06

<https://github.com/nattpana/NattClassFile> <https://nattpana.github.io/ITFnd100-Mod06/>

# Assignment06 - Creating scripts using Functions

## Introduction

In this chapter, I'm expected to learn how to use a parameter, which acts like a bucket that holds inputs and passes them to the output. Additionally, I will learn that functions can be reused and help organize the code. Once defined, functions can be reused multiple times from different parts of the code, making it easier to maintain and update the logic. It also helps implement the "Separation of Concerns" pattern by separating different tasks or concerns into modular units, making the code more focused and easier to understand. When I call functions, I will pass arguments to them. By using the return statement, a function provides output or information by returning a value.

A class differs from a function as it serves as a blueprint or template for creating objects. It defines the structure, behavior, and properties of objects, allowing them to hold data and perform operations.

I also expected to learn about the difference between local and global variables. A local variable is declared inside a function and can only be accessed within that function. On the other hand, a global variable is declared outside any function and can be accessed from any part of the code.

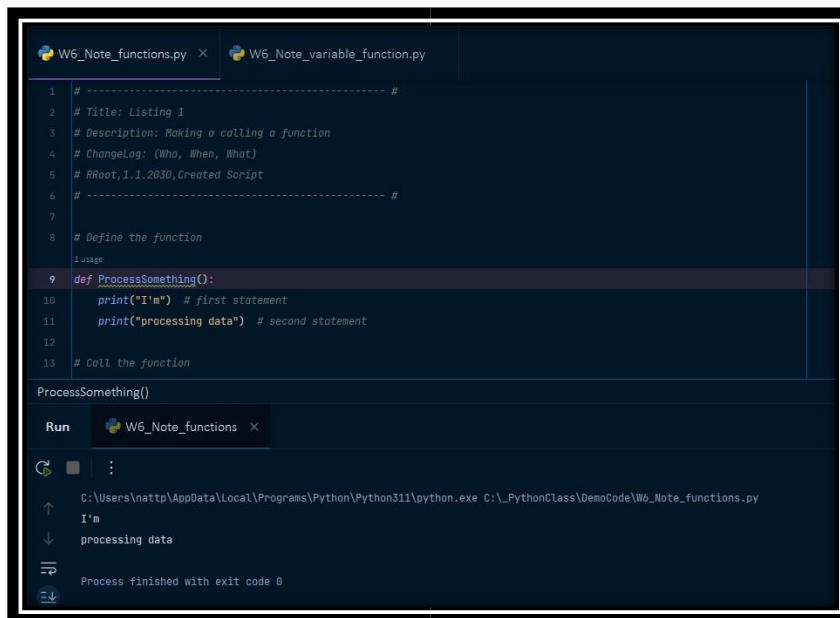
In the PyCharm integrated development environment (IDE) for Python programming, there are various debugging tools available. These include breakpoints, stepping through code, variable inspection, call stack viewing, and error message analysis, which assist in identifying and resolving issues in code.

GitHub enables the publishing of static web content (HTML, CSS, JavaScript) and facilitates sharing projects, documentation, personal websites, and more directly from the repository.

## Practice Functions

## Execute statements by calling functions

After defining the function and when call it, it will execute all processes under that function



The screenshot shows a Python IDE with two tabs: 'W6\_Note\_functions.py' and 'W6\_Note\_variable\_function.py'. The 'W6\_Note\_functions.py' tab is active, displaying the following code:

```
1 # ----- #
2 # Title: Listing 1
3 # Description: Making a calling a function
4 # ChangeLog: (Who, When, What)
5 # RRoot,1.1.2030,Created Script
6 # ----- #
7
8 # Define the function
9 def ProcessSomething():
10     print("I'm") # first statement
11     print("processing data") # second statement
12
13 # Call the function
14 ProcessSomething()
```

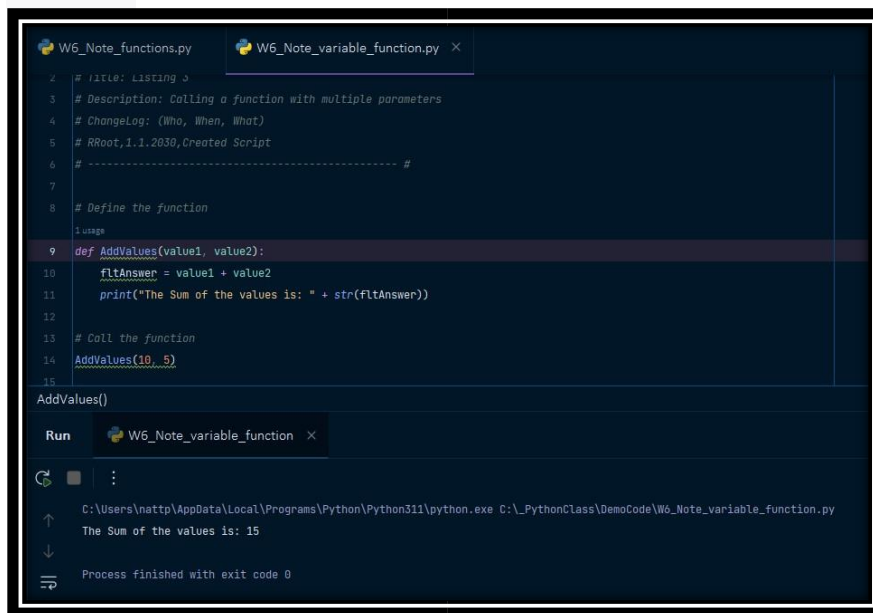
The 'Run' button is highlighted, and the output console shows the following output:

```
C:\Users\nattp\AppData\Local\Programs\Python\Python311\python.exe C:\_PythonClass\DemoCode\W6_Note_functions.py
I'm
processing data
Process finished with exit code 0
```

Figure 1: Example defined and called function.

## Passing arguments in a variable through the functions.

value1 and value2 are the variables that use for passing as an argument to a function called Addvalues



The screenshot shows a Python IDE with two tabs: 'W6\_Note\_functions.py' and 'W6\_Note\_variable\_function.py'. The 'W6\_Note\_variable\_function.py' tab is active, displaying the following code:

```
1 # title: Listing 3
2 # Description: Calling a function with multiple parameters
3 # ChangeLog: (Who, When, What)
4 # RRoot,1.1.2030,Created Script
5 # ----- #
6
7 # Define the function
8 def AddValues(value1, value2):
9     fltAnswer = value1 + value2
10     print("The Sum of the values is: " + str(fltAnswer))
11
12 # Call the function
13 AddValues(10, 5)
```

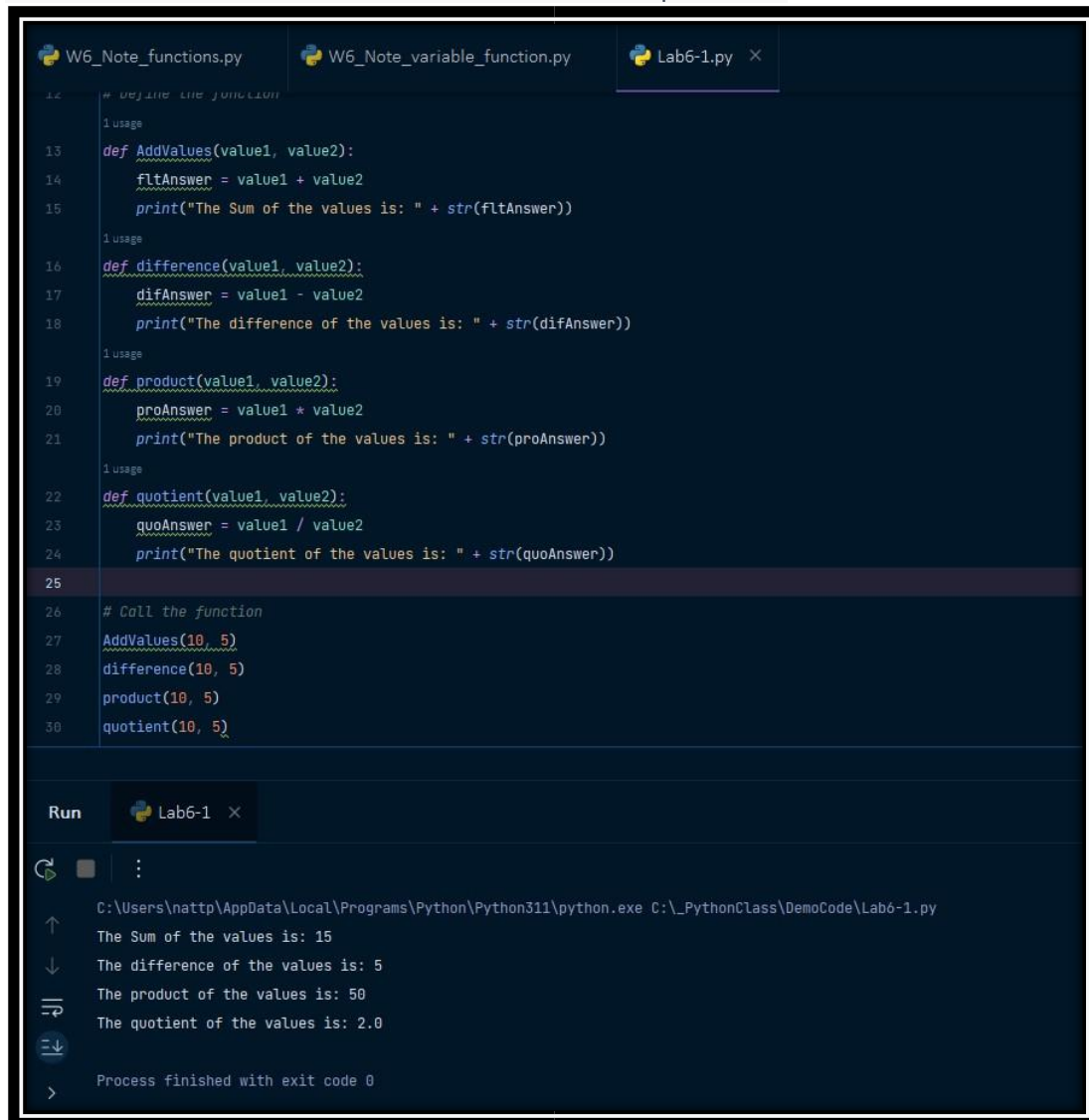
The 'Run' button is highlighted, and the output console shows the following output:

```
C:\Users\nattp\AppData\Local\Programs\Python\Python311\python.exe C:\_PythonClass\DemoCode\W6_Note_variable_function.py
The Sum of the values is: 15
Process finished with exit code 0
```

Figure 2: Example of passing variable through arguments to function.

## Exercise Lab 6 -1

Create and call 4 functions to do the +, -, \*, and / operations.



```
W6_Note_functions.py W6_Note_variable_function.py Lab6-1.py X
12 # Define the function
13 def AddValues(value1, value2):
14     fltAnswer = value1 + value2
15     print("The Sum of the values is: " + str(fltAnswer))
16
17 def difference(value1, value2):
18     difAnswer = value1 - value2
19     print("The difference of the values is: " + str(difAnswer))
20
21 def product(value1, value2):
22     proAnswer = value1 * value2
23     print("The product of the values is: " + str(proAnswer))
24
25 def quotient(value1, value2):
26     quoAnswer = value1 / value2
27     print("The quotient of the values is: " + str(quoAnswer))
28
29 # Call the function
30 AddValues(10, 5)
31 difference(10, 5)
32 product(10, 5)
33 quotient(10, 5)
34
Run Lab6-1 X
C:\Users\natt\AppData\Local\Programs\Python\Python311\python.exe C:\_PythonClass\DemoCode\Lab6-1.py
The Sum of the values is: 15
The difference of the values is: 5
The product of the values is: 50
The quotient of the values is: 2.0
Process finished with exit code 0
```

Figure 3: Lab6-1 call function to do operations and results.

Now I 'm working with the exercise that uses variables as Arguments, in this code is using

"Separation of Concerns" In This example, this exercise starts from

- 1) Data Code part: identify the variable to use in the program
- 2) Processing Code: define the function > process function > print output
- 3) Presentation part separated in

- Input from users to value 1 and value2
- Output from call function Addvalue

```

#-----#
# Title: Listing 04
# Description: Using variables as arguments
# ChangeLog: (Who, When, What)
# RRoot, 01.01.2030, Created Script
#-----#

# -- data code -- #
fltV1 = None # first argument
fltV2 = None # second argument

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

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

# -- Call function -- #
AddValues(fltV1, fltV2)

```

Run W6\_Note\_use variable as arguments

Enter value 2: 5  
The Sum of 2.00 and 5.00  
is: 7.0

Figure 4: Example to get input data to a variable and pass through called function.

When I don't use the return value and try to use the variable inside the function directly the data show as None but when I use the return, I can use that variable from inside the function at the outside function as well.

```

# ChangeLog: (Who, When, What)
# RRoot, 1.1.2030, Created Script
#-----#
# -- processing code -- #
import sys
def AddValues(value1, value2):
    fltAnswer = value1 + value2
    print(fltAnswer) # Doesn't use return
    return fltAnswer

# Call the function and capture the results
fltResults = AddValues(10, 5)
print("The Sum of the values is: " + str(fltResults))

# Or call the function and uses as an expression
print("The Sum of the values is: " + str(AddValues(10, 5)))

AddValues()

```

Run W6\_Note\_return

The Sum of the values is: None  
The Sum of the values is: None

```

# ChangeLog: (Who, When, What)
# RRoot, 1.1.2030, Created Script
#-----#
# -- processing code -- #
import sys
def AddValues(value1, value2):
    fltAnswer = value1 + value2
    print(fltAnswer) # Doesn't use return
    return fltAnswer

# Call the function and capture the results
fltResults = AddValues(10, 5)
print("The Sum of the values is: " + str(fltResults))

# Or call the function and uses as an expression
print("The Sum of the values is: " + str(AddValues(10, 5)))

AddValues()

```

Run W6\_Note\_return

The Sum of the values is: 15  
The Sum of the values is: 15

Figure 5. Show the difference between not call returns and call returns.

I learned that return can work with multiple variables in this program it assigns a new variable to the multiple variable that returns from the function Addvalue

```
4 # RRoot, 01.01.2030, Created Script
5 #-----#
6
7 # -- data code -- #
8 fltV1 = None # first argument
9 fltV2 = None # second argument
10 fltR1 = None # first result of processing
11 fltR2 = None # second result of processing
12 fltR3 = None # third result of processing
13
14 # -- processing code -- #
15 usage
16 def AddValues(value1, value2):
17     fltAnswer = value1 + value2
18     return value1, value2, fltAnswer # pack tuple
19
20 # -- presentation (I/O) code -- #
21 fltV1 = float(input("Enter value 1: "))
22 fltV2 = float(input("Enter value 2: "))
23 fltR1, fltR2, fltR3 = AddValues(fltV1, fltV2) # unpack tuple
24 print("The Sum of %.2f and %.2f is %.2f" % (fltR1, fltR2, fltR3))
25
AddValues()
```

```
5 # RRoot, 01.01.2030, Created Script
6 #-----#
7
8 # -- data code -- #
9 fltV1 = None # first argument
10 fltV2 = None # second argument
11 lstResults = None # list of results for processing
12
13 # -- processing code -- #
14 usage
15 def AddValues(value1, value2):
16     fltAnswer = value1 + value2
17     return [value1, value2, fltAnswer] # create a list
18
19 # -- presentation (I/O) code -- #
20 fltV1 = float(input("Enter value 1: "))
21 fltV2 = float(input("Enter value 2: "))
22 lstResults = AddValues(fltV1, fltV2) # capture the list
23 print("The SUM of %.2f and %.2f is %.2f" %
24       (lstResults[0], lstResults[1], lstResults[2]))
25
AddValues()
```

Run W6\_Note\_Multiple\_Variables X

```
C:\Users\nattp\AppData\Local\Programs\Python\Python311\python.exe C:\_Pyt
Enter value 1: 6
Enter value 2: 9
The Sum of 6.00 and 9.00 is 15.00
```

Run W6\_Note\_Multiple values X

```
C:\Users\nattp\AppData\Local\Programs\Python\Python311\python.exe "C:\_
Enter value 1: 7
Enter value 2: 8
The Sum of 7.00 and 8.00 is 15.00
```

Figure 6. The results of multi-variable returns can use 3 variables to hold the data (on the left) or hold the return value in the list (on the right)

## Exercise Lab 6 -2

I did this exercise by using 2 variables to get input data and using 4 functions to process the code

```

18 # -- processing code -- #
19 1 usage
20 def sumValues(value1, value2):
21     sumAnswer = value1 + value2
22     return [value1, value2, sumAnswer] # create a list
23 1 usage
24 def diffValues(value1, value2):
25     diffAnswer = value1 - value2
26     return [value1, value2, diffAnswer] # create a list
27 1 usage
28 def proValues(value1, value2):
29     proAnswer = value1 * value2
30     return [value1, value2, proAnswer] # create a list
31 1 usage
32 def quoValues(value1, value2):
33     quoAnswer = value1 / value2
34     return [value1, value2, quoAnswer] # create a list
35
36 # -- presentation (I/O) code -- #
37 fltV1 = float(input("Enter value 1: "))
38 fltV2 = float(input("Enter value 2: "))
39
40 sumResults = sumValues(fltV1, fltV2) # capture the list
41 diffResults = diffValues(fltV1, fltV2) # capture the list
42 proResults = proValues(fltV1, fltV2) # capture the list
43 quoResults = quoValues(fltV1, fltV2) # capture the list

```

Figure 7. Lab 6-2 using the list to store and present data when calling function part 1

```

35
36 sumResults = sumValues(fltV1, fltV2) # capture the list
37 diffResults = diffValues(fltV1, fltV2) # capture the list
38 proResults = proValues(fltV1, fltV2) # capture the list
39 quoResults = quoValues(fltV1, fltV2) # capture the list
40
41 print("The Sum of %.2f and %.2f is %.2f" %
42       (sumResults[0], sumResults[1], sumResults[2]))
43
44 print("The Diff of %.2f and %.2f is %.2f" %
45       (diffResults[0], diffResults[1], diffResults[2]))
46
47 print("The Pro of %.2f and %.2f is %.2f" %
48       (proResults[0], proResults[1], proResults[2]))
49
50 print("The Quo of %.2f and %.2f is %.2f" %
51       (quoResults[0], quoResults[1], quoResults[2]))

```

Run Lab 6\_2

```

C:\Users\natt\AppData\Local\Programs\Python\Python311\python.exe "C:\_PythonCla
Enter value 1: 4
Enter value 2: 7
The Sum of 4.00 and 7.00 is 11.00
The Diff of 4.00 and 7.00 is -3.00
The Pro of 4.00 and 7.00 is 28.00
The Quo of 4.00 and 7.00 is 0.57
Process finished with exit code 0

```

Figure 8. Lab 6-2 using the list to store and present data when calling function part 2



## Working with Positional and Name Arguments

It is very convenient to name parameters in arguments.

```
7
8 # -- data code -- #
9 fltV1 = None # first argument
10 fltV2 = None # second argument
11 fltR1 = None # first result of processing
12 fltR2 = None # second result of processing
13 fltR3 = None # third result of processing
14
15 # -- processing code -- #
16 1 usage
17 def AddValues(value1, value2):
18     fltAnswer = value1 + value2
19     return value1, value2, fltAnswer
20
21 # -- presentation (I/O) code -- #
22 fltV1 = float(input("Enter value 1: "))
23 fltV2 = float(input("Enter value 2: "))
24 fltR1, fltR2, fltR3 = AddValues(value1 = fltV1, value2 = fltV2)
25 print("The Sum of %.2f and %.2f is %.2f" % (fltR1, fltR2, fltR3))
26
```

Run W6\_Note\_positional\_name\_arguments

```
C:\Users\nattap\AppData\Local\Programs\Python\Python311\python.exe C:\_Py
Enter value 1: 4
Enter value 2: 6
The Sum of 4.00 and 6.00 is 10.00
Process finished with exit code 0
```

Figure 9. Name parameters in arguments.

```
# -- 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 -- #
1 usage
def AddValues(value1 = 0, value2 = 0):
    fltAnswer = value1 + value2
    return value1, value2, fltAnswer

# -- presentation (I/O) code -- #
fltV1 = float(input("Enter value 1: "))
fltV2 = float(input("Enter value 2: "))
fltR1, fltR2, fltR3 = AddValues(value2 = fltV2)
print("The Sum of %.2f and %.2f is %.2f" % (fltR1, fltR2, fltR3))
```

in w6\_Note\_Default parameter value

```
C:\Users\nattap\AppData\Local\Programs\Python\Python311\python.exe "C:\_Py
Enter value 1: 5
Enter value 2: 8
The Sum of 0.00 and 8.00 is 8.00
Process finished with exit code 0
```

Figure 10. Name parameters in arguments.

I found out in the next step that I can use overload which makes my code simpler by using the function with if\_elif, I can use two variables and identify the variable with the condition of each operation through the arguments.



```
8 # -- processing code -- #
9 # usage
10 def CalcValues(value1 = 0, value2 = 0, operation = '+'):
11     if operation.lower() == '+': fltAnswer = value1 + value2
12     elif operation.lower() == '-': fltAnswer = abs(value1 - value2)
13     elif operation.lower() == '*': fltAnswer = value1 * value2
14     elif operation.lower() == '/': fltAnswer = value1 / value2
15     else: fltAnswer = "Error"
16     return value1, operation, value2, fltAnswer
17
18 # -- presentation (I/O) code -- #
19 print(CalcValues())
20 print(CalcValues(5, 10))
21 print(CalcValues(5, 10, '*'))
22 print(CalcValues(5, 10, '/'))
```

Run W6\_Note\_overload

C:\Users\natp\AppData\Local\Programs\Python\Python311\python.exe C:\\_

(0, '+', 0, 0)

(5, '+', 10, 15)

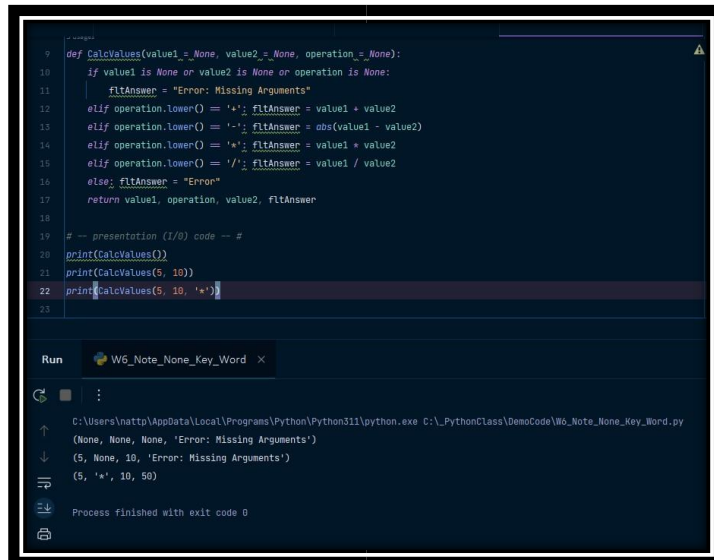
(5, '\*', 10, 50)

(5, '/', 10, 0.5)

Process finished with exit code 0

Figure 11. example of overload

I learned to make the code neater by using the “is None” to prevent any blank input data.



```
9 def CalcValues(value1 = None, value2 = None, operation = None):
10     if value1 is None or value2 is None or operation is None:
11         fltAnswer = "Error: Missing Arguments"
12     elif operation.lower() == '+': fltAnswer = value1 + value2
13     elif operation.lower() == '-': fltAnswer = abs(value1 - value2)
14     elif operation.lower() == '*': fltAnswer = value1 * value2
15     elif operation.lower() == '/': fltAnswer = value1 / value2
16     else: fltAnswer = "Error"
17     return value1, operation, value2, fltAnswer
18
19 # -- presentation (I/O) code -- #
20 print(CalcValues())
21 print(CalcValues(5, 10))
22 print(CalcValues(5, 10, '*'))
```

Run W6\_Note\_None\_Key\_Word

C:\Users\natp\AppData\Local\Programs\Python\Python311\python.exe C:\\_PythonClass\DemoCode\W6\_Note\_None\_Key\_Word.py

(None, None, None, 'Error: Missing Arguments')

(5, None, 10, 'Error: Missing Arguments')

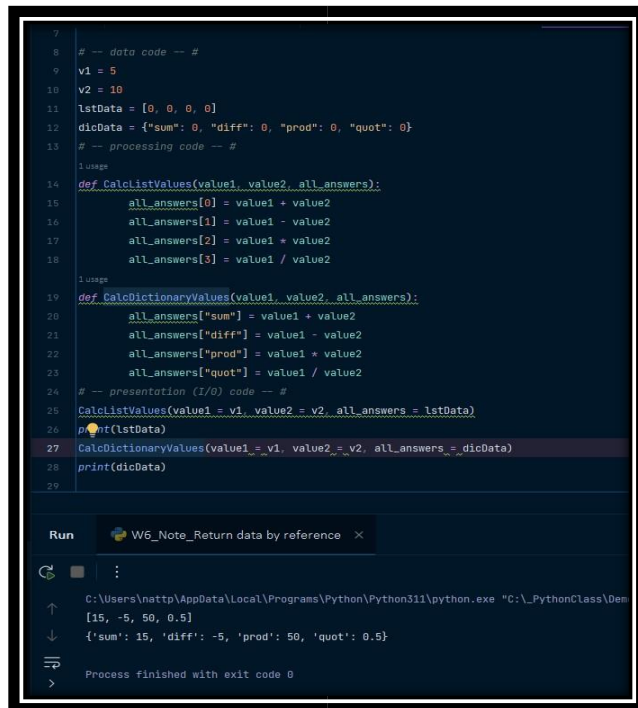
(5, '\*', 10, 50)

Process finished with exit code 0

Figure 11. example of “is None”



I can call functions through both list and dictionary, for list it will be based on the position of data which starts from 0, for the dictionary, I can use short words to name in the arguments and make it easier to understand and follow.



```
7
8 # -- data code -- #
9 v1 = 5
10 v2 = 10
11 lstData = [0, 0, 0, 0]
12 dicData = {"sum": 0, "diff": 0, "prod": 0, "quot": 0}
13 # -- processing code -- #
14
15 usage
16 def CalcListValues(value1, value2, all_answers):
17     all_answers[0] = value1 + value2
18     all_answers[1] = value1 - value2
19     all_answers[2] = value1 * value2
20     all_answers[3] = value1 / value2
21
22 usage
23 def CalcDictionaryValues(value1, value2, all_answers):
24     all_answers["sum"] = value1 + value2
25     all_answers["diff"] = value1 - value2
26     all_answers["prod"] = value1 * value2
27     all_answers["quot"] = value1 / value2
28
29 # -- presentation (I/O) code -- #
30
31 CalcListValues(value1 = v1, value2 = v2, all_answers = lstData)
32 print(lstData)
33 CalcDictionaryValues(value1 = v1, value2 = v2, all_answers = dicData)
34 print(dicData)
35
36 Run V6_Note_Return data by reference X
37
38 C:\Users\natp\AppData\Local\Programs\Python\Python311\python.exe "C:\_PythonClass\Dem
39 [15, -5, 50, 0.5]
40 {'sum': 15, 'diff': -5, 'prod': 50, 'quot': 0.5}
41
42 Process finished with exit code 0
43 >
```

Figure 12. Compare results calling a function on a list and dictionary

I learned that global parameters should be assigned at the top of the program so that they can be accessed and used throughout the entire program. These global parameters have a broader scope and can be utilized in any part of the program. The global parameter will take priority over all local parameter that has the same name, which calls the shadowing global variable. For better results, we should not use global variation inside the function.

On the other hand, local parameters are assigned within specific functions or blocks of code and are only accessible within that specific function or block. They have a narrower scope and are used exclusively for that internal function. This practice helps ensure proper organization and separation of variables in a program, making it easier to understand and maintain.

Docstrings, short for documentation strings, are used in programming languages, particularly in Python, to provide a description and documentation for functions, modules, classes, or methods. They are enclosed within triple quotes (""") and are placed at the beginning of the entity they are documenting.

```

6 # -----#
7
8 # -- data code -- #
9 # Note: Variables declared in the body of the script are "Global"
10 v1 = 10 # first argument
11 v2 = 5 # second argument
12 gAnswer = None # result of processing
13
14 # -- processing code -- #
15
16 def AddValues(value1, value2):
17     global gAnswer # This refers to the "global" variable
18     gAnswer = value1 + value2
19     answer = value1 + value2 # This is a "local" variable!
20     return answer
21
22 # -- presentation (I/O) code -- #
23 AddValues(v1, v2)
24 print('Global = ', gAnswer)
25 print('Local = ', AddValues(v1, v2))
26

```

Run W5\_Note\_Global variable x

C:\Users\natt\AppData\Local\Programs\Python\Python311\python.exe "C:\PythonClass\DemoCode\W5\_Note\_Global variable.py"

Global = 15  
Local = 15

Process finished with exit code 0

Figure 13. Compare global parameter and local parameter

```

2 # Title: Listing 15
3 # Description: Global variable Shadowing
4 # ChangeLog: (Who, When, What)
5 # RRoot, 01.01.2030, Created Script
6 # -----#
7
8 # -- data code -- #
9 v1 = 10 # first argument
10 v2 = 5 # second argument
11 answer = None # result of processing???
12
13 # -- processing code -- #
14
15 def AddValues(value1, value2):
16     answer = value1 + value2 # answer "shadowed" the global variable
17
18 # -- presentation (I/O) code -- #
19 AddValues(v1, v2)
20 print('Global = ', answer)
21

```

AddValues()

Run W6\_Note\_Shadow variable x

C:\Users\natt\AppData\Local\Programs\Python\Python311\python.exe "C:\PythonClass\DemoCode\W6\_Note\_Shadow variable.py"

Global = None

Process finished with exit code 0

Figure 14. Shadowing global parameter; when it is the same name

*The global parameter takes priority over the local parameter*

## Lab 6\_3

Practice using the functions from the Mathprocessor class to perform addition (+), subtraction (-), multiplication (\*), and division (/). I noticed that we don't need to add any operations as they are already predefined within the Mathprocessor class. To call the function, you only need to pass value1 and value2

as arguments. The code format will have Mathprocessor in front of the function name and use "." to separate them.

```
15 # Root, 01.01.2020, Created Script
16 # -----#
17 LAB 6_3
18 # Create class called MathProcessor
19
20 class MathProcessor:
21     # 3. Add a function called AddValues to process the sum of two values and return the result as a float.
22     def AddValues(self, value1=0.0, value2=0.0):
23         return float(value1 + value2)
24
25     # 4. Add a function called SubtractValues to process the difference of two values and return the result as a float.
26     def SubtractValues(self, value1=0.0, value2=0.0):
27         return float(value1 - value2)
28
29     # 5. Add a function called MultiplyValues to process the product of two values and return the result as a float.
30     def MultiplyValues(self, value1=0.0, value2=0.0):
31         return float(value1 * value2)
32
33     # 6. Add a function called DivideValues to process the quotient of two values and return the result as a float.
34     def DivideValues(self, value1=0.0, value2=0.0):
35         return float(value1 / value2)
36
37 # 7. Add code to capture two values from a user.
38 print("Please choose the operation:")
39 print("Choice 1: AddValues")
40 print("Choice 2: SubtractValues")
41 print("Choice 3: MultiplyValues")
42 print("Choice 4: DivideValues")
```

Figure 15. Lab 6\_3 Lean to use Mathprocessor class part 1

```
44 choice = int(input("Enter your choice: "))
45 value1 = float(input("Please enter the first value: "))
46 value2 = float(input("Please enter the second value: "))
47
48 math_processor = MathProcessor()
49
50 if choice == 1:
51     result = math_processor.AddValues(value1, value2)
52 elif choice == 2:
53     result = math_processor.SubtractValues(value1, value2)
54 elif choice == 3:
55     result = math_processor.MultiplyValues(value1, value2)
56 elif choice == 4:
57     result = math_processor.DivideValues(value1, value2)
58 else:
59     result = "Invalid choice"
60
61 print("Result:", result)
```

Run Lab 6\_3

C:\Users\natt\AppData\Local\Programs\Python\Python311\python.exe "C:\PythonClass\DemoCode\Lab 6\_3.py"

Please choose the operation:  
Choice 1: AddValues  
Choice 2: SubtractValues  
Choice 3: MultiplyValues  
Choice 4: DivideValues  
Enter your choice: 1  
Please enter the first value: 4  
Please enter the second value: 5  
Result: 9.0

Figure 16. Lab 6\_3 Lean to use Mathprocessor class part 2

## Debug

When I click to assign the breakpoints in front of the line, when I ran until that break line it will stop and show the details parameter at that step.

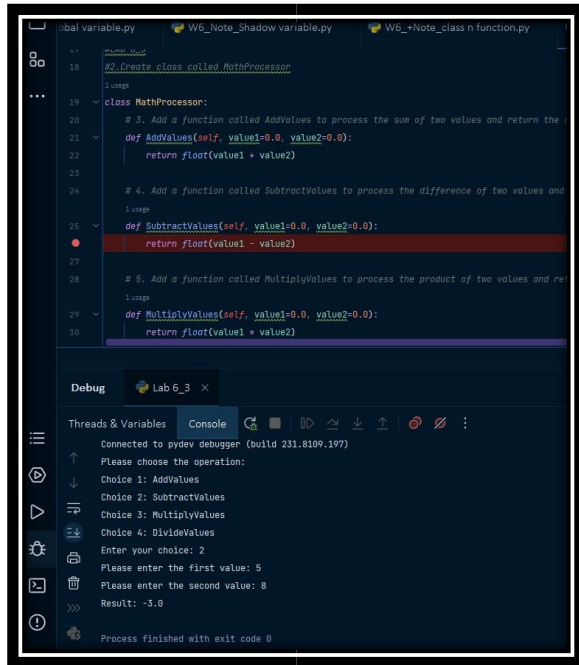


Figure 17. Example of Debug Part 1

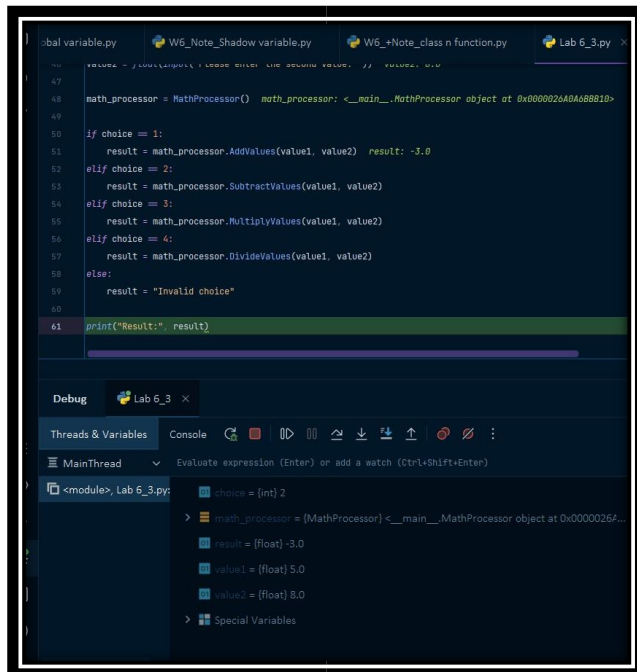


Figure 18. Example of Debug Part 2

## GitHub

I have created a folder and a new file, I have an issue when trying to do the configuration, it's not as same as in the exercise, and I'm not able to find the step of drop-down source

<https://nattpana.github.io/ITFnd100-Mod06/>

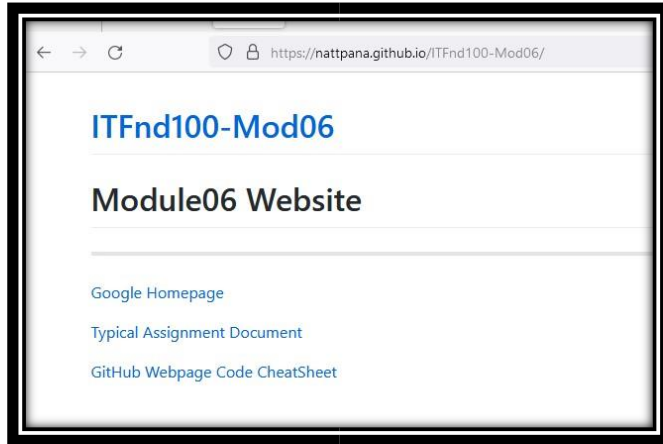


Figure 19. Example of building Github

## Assignment 06

This assignment has the precoding and the focus is to add some coding to complete the pre-coding. Assignment06 wants me to be able to use the choice to be able to 1) Add a new Task 2) Remove an existing task 3) save Data in the file (name ToDoFile.txt) into the Python dictionary.

```
1 # ----- #
2 # 📌 file: Assignment 06_Natt #
3 # Description: Working with functions in a class,
4 #             When the program starts, load each "row" of data
5 #             in "ToDoFile.txt" into a python Dictionary.
6 #             Add each dictionary "row" to a python List "table"
7 # ChangeLog (Who,When,What):(Natta Panapornsirikun , 05/24/2023, Filling the missing code)
8 # RRoot,1.1.2030,Created started script
9 # <Natta Panapornsirikun>,<05/24/2023>,<Modified code to complete assignment 06>
10 # ----- #
11
12 # Data ----- #
13 # Declare variables and constants
14 ToDoFile_str = "ToDoFile.txt" # The name of the data file
15 row_dic = {} # A row of data separated into elements of a dictionary {Task,Priority}
16 table_list = [] # A list that acts as a 'table' of rows
17 choice_str = "" # Captures the user option selection
18
19
20 # Processing ----- #
21 class Processor:
22     """ Performs Processing tasks """
23
24     @staticmethod
25     def read_data_from_file(file_name, list_of_rows):
26         """ Reads data from a file into a list of dictionary rows
27             :param file_name: (string) with name of file:
28             :param list_of_rows: (list) you want filled with file data:
29             :return: (list) of dictionary rows
30         """
31
```

Figure20. Assignment 06 code PART1

```
28 :param list_of_rows: (list) you want filled with file data:
29 :return: (list) of dictionary rows
30 """
31
32 list_of_rows.clear() # clear current data
33 try:
34     file = open(file_name, "r") # load each "row" of data
35     for line in file:
36         task, priority = line.strip().split(",")
37         row = {"Task": task, "Priority": priority}
38         list_of_rows.append(row)
39     file.close()
40 except FileNotFoundError:
41     print("File not found. Creating a new file.")
42     return list_of_rows
43
44 @staticmethod
45 def add_data_to_list(task, priority, list_of_rows):
46     """ Adds data to a list of dictionary rows
47     :param task: (string) with name of task:
48     :param priority: (string) with name of priority:
49     :param list_of_rows: (list) you want to add more data to:
50     :return: (list) of dictionary rows
51     """
52     row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
53     list_of_rows.append(row)
54     return list_of_rows
55
56 @staticmethod
57 def remove_data_from_list(task, list_of_rows):
58     """ Removes data from a list of dictionary rows
59
```

Figure21. Assignment 06 code PART2

```
59
60 :param task: (string) with name of task:
61 :param list_of_rows: (list) you want filled with file data:
62 :return: (list) of dictionary rows
63 """
64 list_of_rows = [row for row in list_of_rows if row["Task"].lower() != task.lower()]
65 return list_of_rows
66
67 @staticmethod
68 def write_data_to_file(file_name, list_of_rows):
69     """ Writes data from a list of dictionary rows to a File
70
71     :param file_name: (string) with name of file:
72     :param list_of_rows: (list) you want filled with file data:
73     :return: (list) of dictionary rows
74     """
75     file = open(file_name, "w") # load each "row" of data
76     for row in list_of_rows:
77         task = row["Task"]
78         priority = row["Priority"]
79         file.write(f"{task},{priority}\n")
80     file.close()
81     return list_of_rows
82
83
84 # Presentation (Input/Output) ----- #
85
```

Figure22. Assignment 06 code PART3

```

86
87 class IO:
88     """ Performs Input and Output tasks """
89
90     @staticmethod
91     def output_menu_tasks():
92         """ Display a menu of choices to the user
93
94         :return: nothing
95         """
96         print('''
97         Menu of Options
98         1) Please add a new Task
99         2) Please remove an existing Task
100        3) Please Save Data to File
101        4) Exit Program
102        ''')
103        print() # Add an extra line for looks
104
105     @staticmethod
106     def input_menu_choice():
107         """ Gets the menu choice from a user
108
109         :return: string
110         """
111         choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()
112         print() # Add an extra line for looks
113         return choice
114
115     @staticmethod
116     def output_current_tasks_in_list(list_of_rows):

```

Figure23. Assignment 06 code PART4

```

115     @staticmethod
116     def output_current_tasks_in_list(list_of_rows):
117         """ Shows the current Tasks in the list of dictionaries rows
118
119         :param list_of_rows: (list) of rows you want to display
120         :return: nothing
121         """
122         print("***** The current tasks ToDo are: *****")
123         for row in list_of_rows:
124             print(row["Task"] + " (" + row["Priority"] + ")")
125         print("*****")
126         print() # Add an extra line for looks
127
128     @staticmethod
129     def input_new_task_and_priority():
130         """ Gets task and priority values to be added to the list
131
132         :return: (string, string) with task and priority
133         """
134         task = str(input("Enter Task? - ")).strip()
135         priority = str(input("Enter priority? - ")).strip()
136         return task, priority
137
138     @staticmethod
139     def input_task_to_remove():
140         """ Gets the task name to be removed from the list
141
142         :return: (string) with task
143         """
144         task = str(input("Enter Task you want to delete: - ")).strip()
145         return task
146

```

Figure24. Assignment 06 code PART5



```
148 # Main Body of Script ----- #
149
150 # Step 1 - When the program starts, Load data from ToDoFile_str.
151 Processor.read_data_from_file(ToDoFile_str, table_lst) # read file data
152
153 # Step 2 - Display a menu of choices to the user
154 while True:
155     # Step 3 Show current data
156     IO.output_current_tasks_in_list(list_of_rows=table_lst) # Show current data in the list/table
157     IO.output_menu_tasks() # Shows menu
158     choice_str = IO.input_menu_choice() # Get menu option
159
160     # Step 4 - Process user's menu choice
161     if choice_str.strip() == '1': # Add a new Task
162         task, priority = IO.input_new_task_and_priority()
163         table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
164         continue # to show the menu
165
166     elif choice_str == '2': # Remove an existing Task
167         task = IO.input_task_to_remove()
168         table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
169         continue # to show the menu
170
171     elif choice_str == '3': # Save Data to File
172         table_lst = Processor.write_data_to_file(file_name=ToDoFile_str, list_of_rows=table_lst)
173         print("Data Saved!")
174         continue # to show the menu
175
176     elif choice_str == '4': # Exit Program
177         print("Goodbye!")
```

Figure25. Assignment 06 code PART6

```
C:\Users\nattp\AppData\Local\Programs\Python\Python311\python.exe
File not found. Creating a new file.
***** The current tasks ToDo are: *****
*****

Menu of Options
1) Please add a new Task
2) Please remove an existing Task
3) Please Save Data to File
4) Exit Program

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

Enter Task? - swim
Enter priority? - 2
***** The current tasks ToDo are: *****
swim (2)
*****

Menu of Options
1) Please add a new Task
2) Please remove an existing Task
3) Please Save Data to File
4) Exit Program
```

Figure26. Assignment 06 Results PART1

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

Enter Task? - read
Enter priority? - 1
***** The current tasks ToDo are: *****
swim (2)
read (1)
*****

Menu of Options
1) Please add a new Task
2) Please remove an existing Task
3) Please Save Data to File
4) Exit Program

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

Enter Task? - run
Enter priority? - 3
***** The current tasks ToDo are: *****
swim (2)
read (1)
run (3)
*****
```

Figure27. Assignment 06 Results PART2

```
Menu of Options
1) Please add a new Task
2) Please remove an existing Task
3) Please Save Data to File
4) Exit Program

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

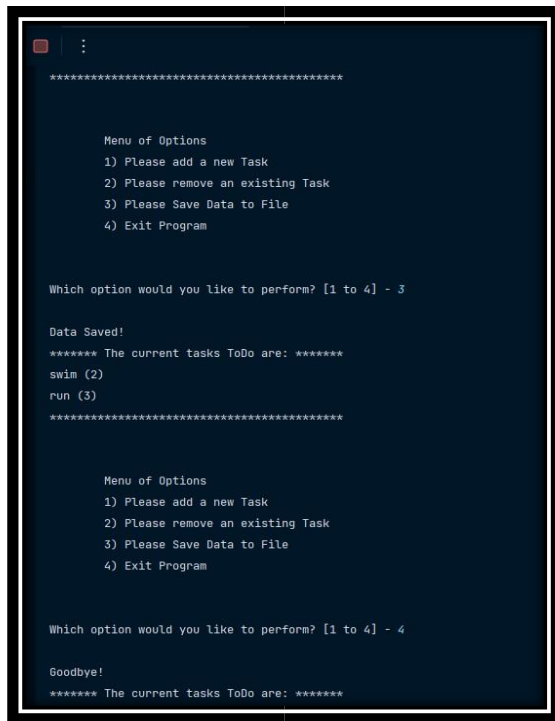
Enter Task you want to delete: - 2
***** The current tasks ToDo are: *****
swim (2)
read (1)
run (3)
*****

Menu of Options
1) Please add a new Task
2) Please remove an existing Task
3) Please Save Data to File
4) Exit Program

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

Enter Task you want to delete: - read
***** The current tasks ToDo are: *****
swim (2)
run (3)
```

Figure28. Assignment 06 Results PART3



```
*****

Menu of Options
1) Please add a new Task
2) Please remove an existing Task
3) Please Save Data to File
4) Exit Program

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

Data Saved!
***** The current tasks ToDo are: *****
swim (2)
run (3)
*****

Menu of Options
1) Please add a new Task
2) Please remove an existing Task
3) Please Save Data to File
4) Exit Program

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

Goodbye!
***** The current tasks ToDo are: *****
```

Figure29. Assignment 06 Results PART4

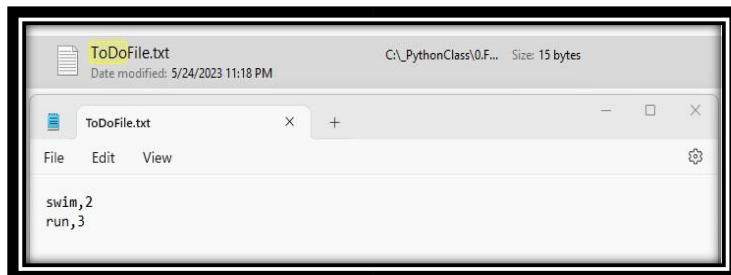


Figure30. Assignment 06 Results PART5 in To do list text file

```
C:\WINDOWS\system32\cmd. x + v
Natta.py"
***** The current tasks ToDo are: *****
swim (2)
run (3)
*****

Menu of Options
1) Please add a new Task
2) Please remove an existing Task
3) Please Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4] - 1
Enter Task? - Read
Enter priority? - 1
***** The current tasks ToDo are: *****
swim (2)
run (3)
Read (1)
*****

Menu of Options
1) Please add a new Task
2) Please remove an existing Task
3) Please Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4] - 1
Enter Task? - SWIM
Enter priority? - 2
***** The current tasks ToDo are: *****
swim (2)
run (3)
Read (1)
SWIM (2)
*****

Menu of Options
1) Please add a new Task
2) Please remove an existing Task
3) Please Save Data to File
4) Exit Program
```

Figure31. Assignment 06 Results in bat file

## Summary

I have learned about functions, parameters, return statements, classes, local and global variables, debugging tools, and GitHub. These concepts will help me organize my code, promote reusability, handle inputs and outputs, define object structures, debug my code, and facilitate project management and sharing.