



Name : Huzaifa Syed

Roll No. F24-611

Program : BSSE

Semester/section : 2nd C

Subject : OOP

Session : Fall 2024

Submitted To : Jamal Abdul Ahad

Department of Computer science

Abbottabad University of Science & Technology

Part 1: Introduction to Functions :

A function is a block of reusable code that performs a specific task. Functions help in organizing code and improving reusability.

Defining a Function :

A function is defined using the “**def**” keyword.

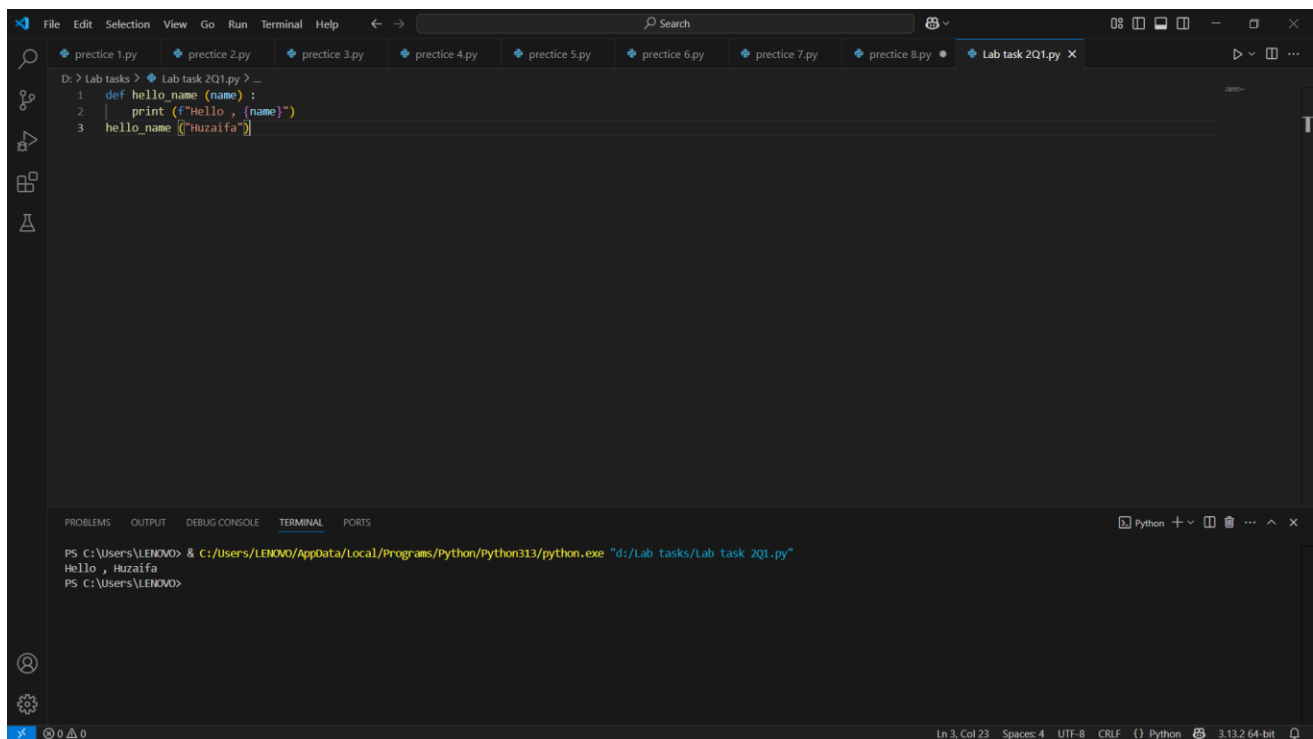
Example :

```
def greet():  
    print("Hello, World!")  
greet() # Calling the function
```

Question No. 1 :

Write a function `hello_name` that takes a name as an argument and prints "Hello, !".

Code :



The screenshot shows a code editor with a file explorer on the left displaying several Python files. The active file, 'Lab task 2Q1.py', contains the following code:

```
D:\> Lab tasks > Lab task 2Q1.py > ...  
1 def hello_name (name) :  
2     print (f"Hello , {name}")  
3 hello_name ("Huzaiifa")
```

Below the code editor is a terminal window showing the execution of the script:

```
PS C:\Users\LENOVO> & C:\Users\LENOVO\AppData\Local\Programs\Python\Python313\python.exe "d:/Lab tasks/Lab task 2Q1.py"  
Hello , Huzaiifa  
PS C:\Users\LENOVO>
```

The status bar at the bottom indicates the current position is Line 3, Column 23, with 4 spaces, using UTF-8 encoding and CRLF line endings for a Python 3.13.2 64-bit file.

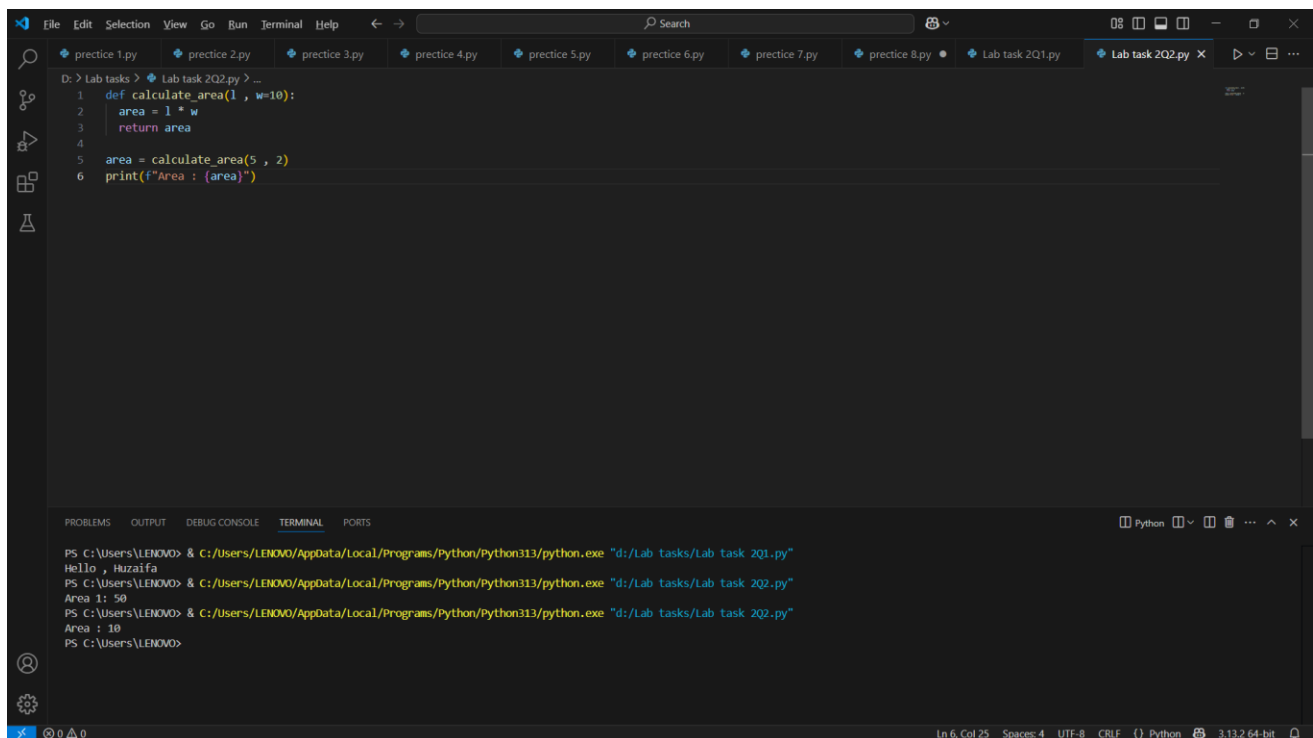
Part 2: Function Arguments :

Functions can have parameters that receive values when called.

Question No. 2 :

Write a function `calculate_area` that takes length and width as arguments and returns the area of a rectangle. The width should have a default value of 10.

Code :



```
D:\> Lab tasks > Lab task 2Q2.py > ...
1  def calculate_area(l , w=10):
2      area = l * w
3      return area
4
5  area = calculate_area(5 , 2)
6  print(f"Area : {area}")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
Python

PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task 2Q1.py"
Hello , Huzaiifa
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task 2Q2.py"
Area : 50
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task 2Q2.py"
Area : 10
PS C:\Users\LENOVO>
```

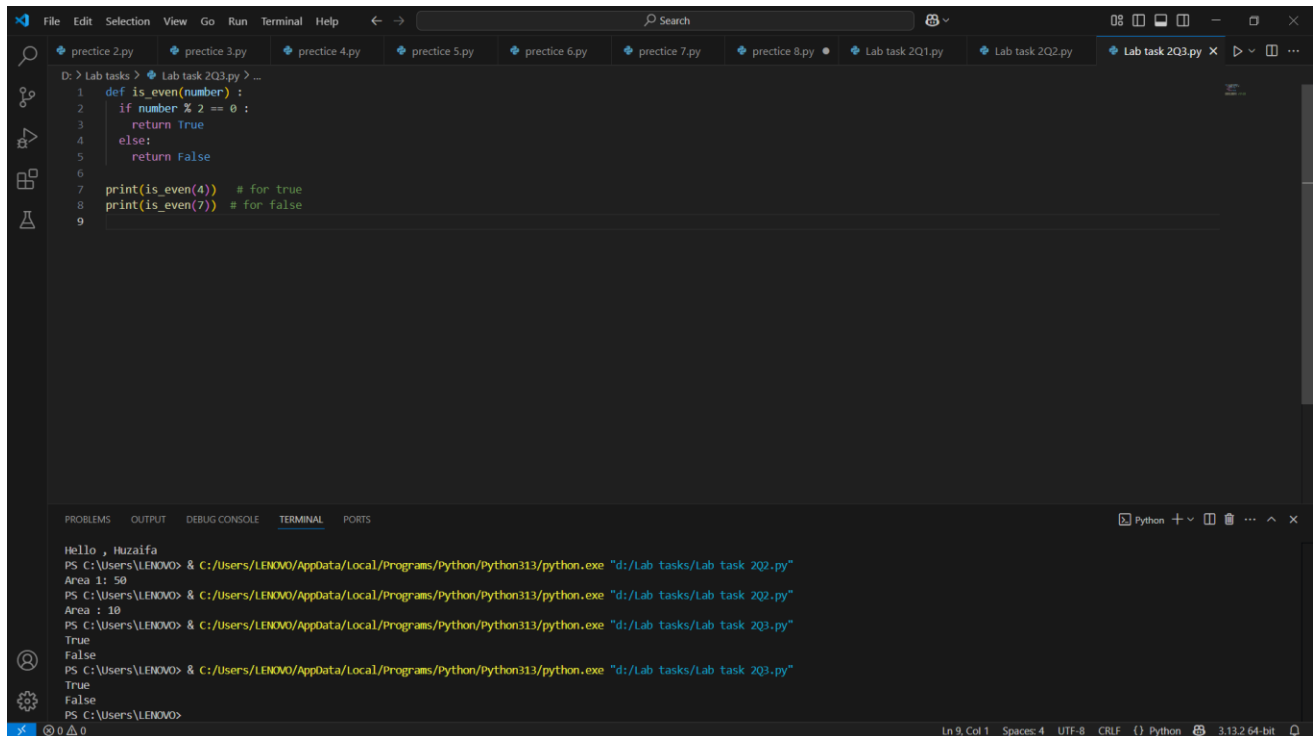
Part 3: Return Values :

A function can return a value using the “**return**” keyword.

Question No. 3 :

Write a function `is_even` that returns `True` if a given number is even and `False` otherwise.

Code :



```
D:\> Lab tasks > Lab task Q3.py > ...
1 def is_even(number) :
2     if number % 2 == 0 :
3         return True
4     else:
5         return False
6
7 print(is_even(4)) # for true
8 print(is_even(7)) # for false
9
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Hello , Huzaifa
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task Q2.py"
Area 1: 50
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task Q2.py"
Area : 10
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task Q3.py"
True
False
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task Q3.py"
True
False
PS C:\Users\LENOVO>
```

Ln 9, Col 1 Spaces: 4 UTF-8 CRLF Python 3.13.2 64-bit

Part 4: Variable Scope :

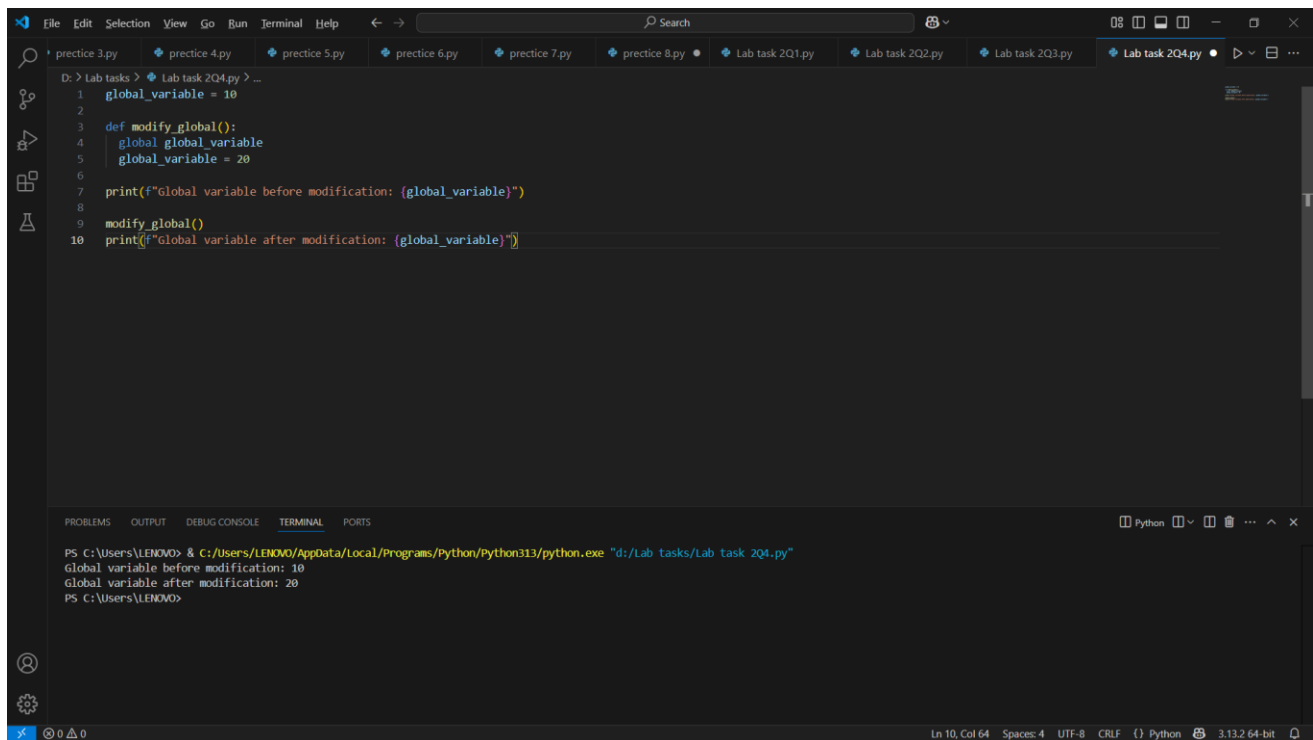
Local and Global Variables ...

Variables defined inside a function are local, while those outside are global.

Question No. 4 :

Create a function that modifies a global variable inside a function using the global keyword.

Code :



```
D:\> Lab tasks > Lab task 2Q4.py > ...
1 global_variable = 10
2
3 def modify_global():
4     global global_variable
5     global_variable = 20
6
7 print(f"Global variable before modification: {global_variable}")
8
9 modify_global()
10 print(f"Global variable after modification: {global_variable}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task 2Q4.py"
Global variable before modification: 10
Global variable after modification: 20
PS C:\Users\LENOVO>
```

Ln 10, Col 64 Spaces: 4 UTF-8 CRLF {} Python 3.13.2 64-bit

Part 5: Recursion

A function can call itself, known as recursion.

Question No. 5

Write a recursive function "fibonacci(n)" that returns the nth Fibonacci number.

Code :

The screenshot shows a VS Code editor window with a Python file named 'Lab task 2Q5.py'. The code defines a Fibonacci function and prints the result for n=0. The terminal shows the command to run the script and its output.

```

File Edit Selection View Go Run Terminal Help
D:\> Lab tasks > Lab task 2Q5.py > ...
1 def fibonacci(n):
2
3     if n <= 1:
4         return n
5     else:
6         return fibonacci(n - 1) + fibonacci(n - 2)
7
8
9 print(fibonacci(0))

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\LENOVO> & C:\Users\LENOVO\AppData\Local\Programs\python\python313/python.exe "d:/Lab tasks/Lab task 2Q5.py"
0
PS C:\Users\LENOVO>

```

Part 6: Lambda Functions

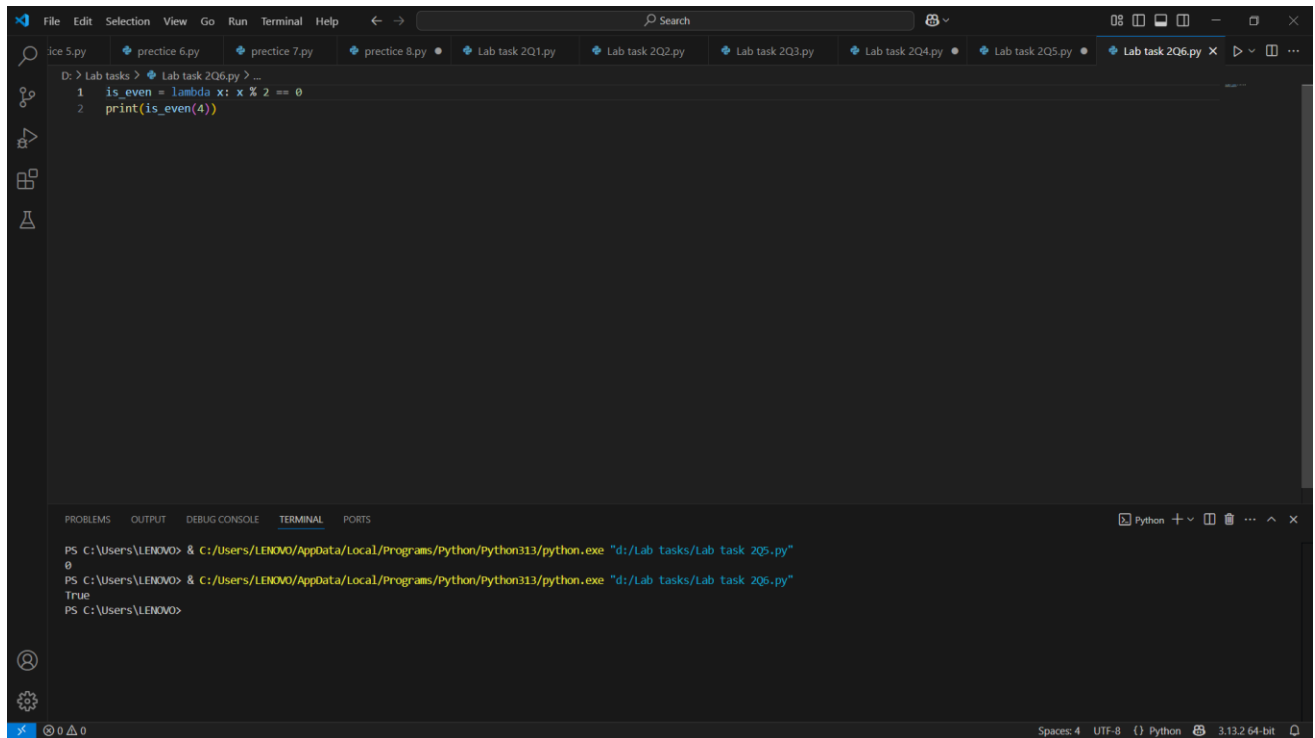
Lambda functions are anonymous functions in Python.

```
square = lambda x: x * x  
print(square(5))
```

Question No. 6

Write a lambda function to check if a number is even.

Code :



The screenshot shows a Python IDE with a dark theme. The top toolbar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The editor window displays a file named 'Lab task 2Q6.py' with the following code:

```
D:\> Lab tasks > Lab task 2Q6.py > ...  
1 is_even = lambda x: x % 2 == 0  
2 print(is_even(4))
```

The bottom panel shows the TERMINAL output:

```
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task 2Q5.py"  
0  
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task 2Q6.py"  
True  
PS C:\Users\LENOVO>
```

The status bar at the bottom indicates 'Spaces: 4', 'UTF-8', 'Python', and '3.13.2 64-bit'.

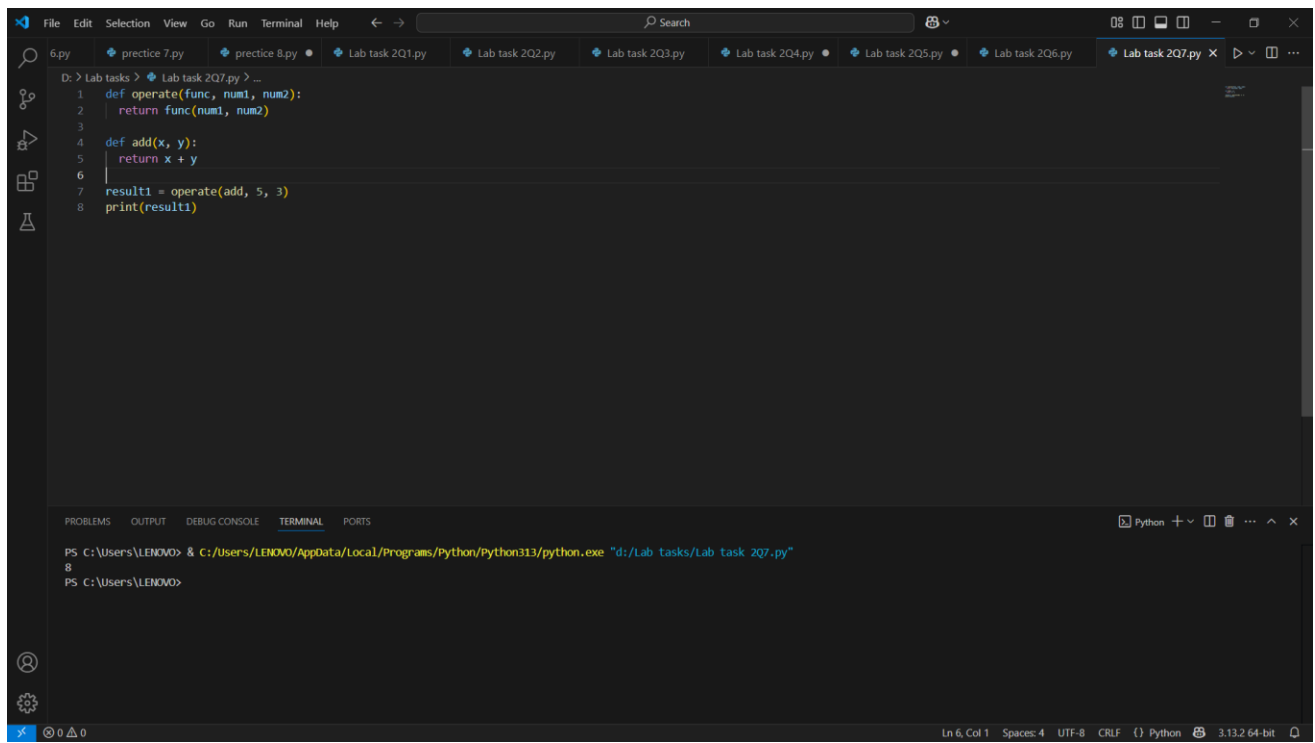
Part 7: Function as Arguments

Functions can be passed as arguments to other functions.

Question No. 7

Write a function operate that takes another function as an argument and applies it to two numbers.

Code :



```
D:\> Lab tasks > Lab task 2Q7.py > ...
1 def operate(func, num1, num2):
2     return func(num1, num2)
3
4 def add(x, y):
5     return x + y
6
7 result1 = operate(add, 5, 3)
8 print(result1)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task 2Q7.py"
8
PS C:\Users\LENOVO>
```

Ln 6, Col 1 Spaces: 4 UTF-8 CRLF Python 3.13.2 64-bit

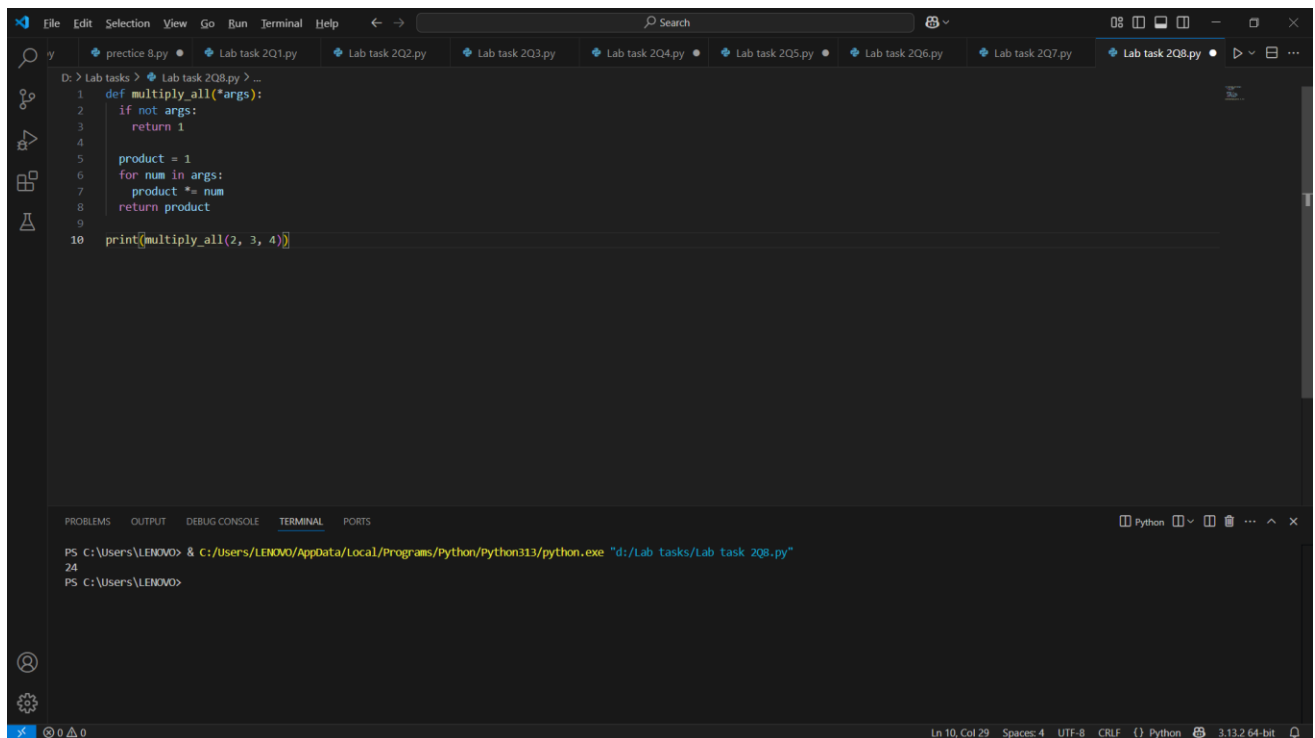
Part 8: *args and **kwargs :

Using *args for Variable-Length Arguments.

Question No. 8

Write a function multiply_all that accepts multiple arguments using *args and returns their product.

Code :



The screenshot shows a Visual Studio Code editor window with a Python file named 'Lab task 2Q8.py'. The code defines a function 'multiply_all' that takes a variable number of arguments (*args). It checks if there are any arguments; if not, it returns 1. Otherwise, it initializes a 'product' variable to 1 and iterates through each number in 'args', multiplying them together. Finally, it prints the result of 'multiply_all(2, 3, 4)'. Below the editor, the terminal shows the command to run the script and the output '24'.

```
D:\> Lab tasks > Lab task 2Q8.py > ...
1 def multiply_all(*args):
2     if not args:
3         return 1
4
5     product = 1
6     for num in args:
7         product *= num
8     return product
9
10 print(multiply_all(2, 3, 4))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\LENOVO> & C:\Users\LENOVO\AppData\Local\Programs\Python\Python313\python.exe "d:/Lab_tasks/Lab_task_2Q8.py"

24

PS C:\Users\LENOVO>

Ln 10, Col 29 Spaces: 4 UTF-8 CRLF Python 3.13.2 64-bit

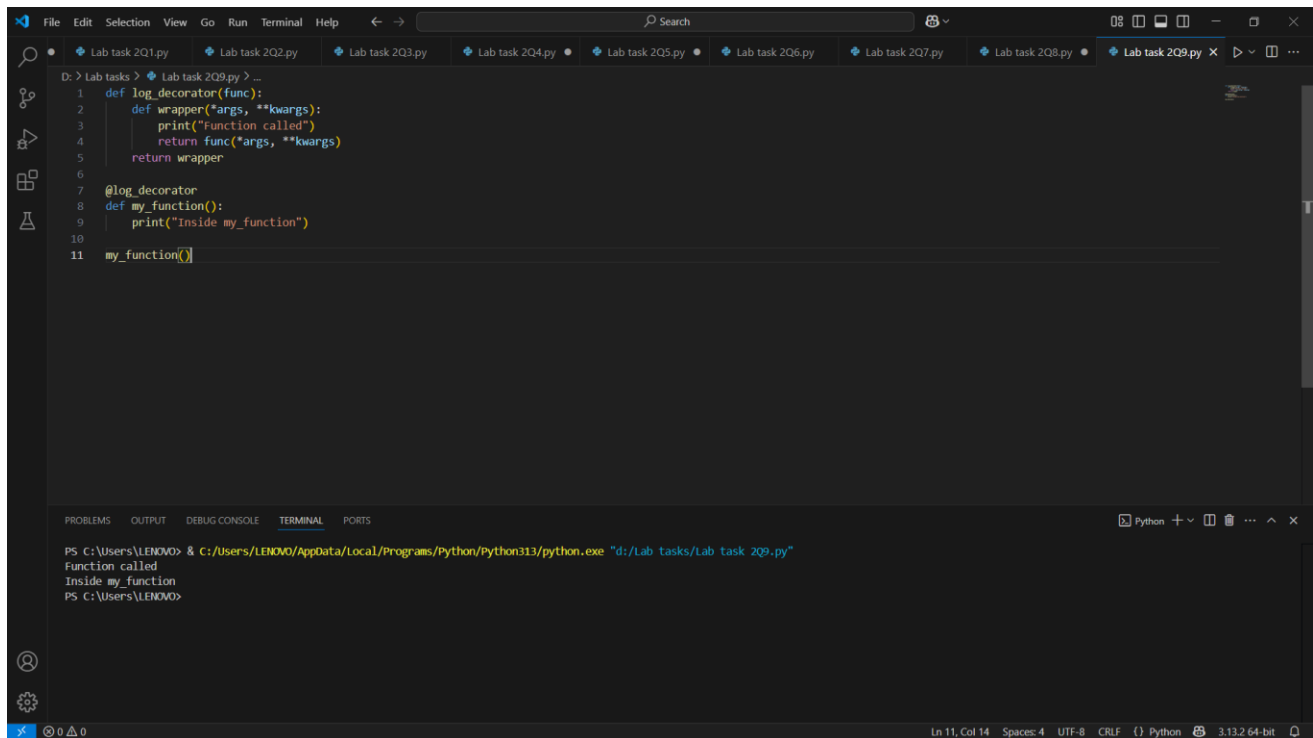
Part 9: Function Decorators

Decorators modify the behavior of a function

Question No. 9

Write a decorator `log_decorator` that prints "Function called" before executing the function.

Code : `"""took help from ai."""`



```
D:\> Lab tasks > Lab task 2Q9.py > ...
1 def log_decorator(func):
2     def wrapper(*args, **kwargs):
3         print("Function called")
4         return func(*args, **kwargs)
5     return wrapper
6
7 @log_decorator
8 def my_function():
9     print("Inside my_function")
10
11 my_function()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\LENOVO> & c:/Users/LENOVO/AppData/Local/Programs/Python/Python313/python.exe "d:/Lab tasks/Lab task 2Q9.py"
Function called
Inside my_function
PS C:\Users\LENOVO>
```

Ln 11, Col 14 Spaces: 4 UTF-8 CRLF Python 3.13.2 64-bit