17/09/25 – Custom Functions in Python

🔹 What is a Function?

A **function** in Python is a block of organized, reusable code that is used to perform a single, related action. Functions help in dividing a large program into smaller, manageable, and reusable blocks of code.

👉 **Why use functions?**

1. **Reusability** – Write once, use many times.
2. **Readability** – Makes the program cleaner and easier to understand.
3. **Maintainability** – Easy to update or debug.
4. **Modularity** – Divides the program into logical parts.

🔹 Defining a Function in Python

The `def` keyword is used.

Syntax:

```
def function_name(parameters):
    """Optional docstring: explains what the function does"""
    # body of the function
    return result
```

🔹 Types of Functions

1. **Built-in Functions**: Already available in Python (e.g., `print()`, `len()`, `type()`).
2. **User-defined Functions**: Created by the programmer using `def`.

🔹 Examples

Example 1: Function without arguments

🔹 Theory:

- A **function without arguments** does not take any input values.
- It simply performs a task whenever it is called.
- Such functions are useful when the output does not depend on user-provided data.

```
def welcome():
    print("Hello, welcome to Python functions!")

welcome()
```

Output:

```
Hello, welcome to Python functions!
```

Example 2: Function with arguments

🔹 Theory:

- Functions can take **arguments (parameters)**.
- Arguments allow us to pass input values into the function.
- The function can then use those inputs to perform tasks.

```
def greet(name):
    print("Hello", name, "!")

greet("Alice")
greet("Bob")
```

Output:

```
Hello Alice !
Hello Bob !
```

Example 3: Function with default argument

🔹 Theory:

- A **default argument** is a value that is used if the user does not provide one.
- This makes the function more flexible.
- If the user gives a value, it overrides the default.

```
def power(base, exp=2):   # default exponent is 2
    return base ** exp

print(power(5))       # 25
print(power(5, 3))    # 125
```

Output:

```
25
125
```

Example 4: Function returning multiple values

🔹 Theory:

- A function in Python can return **more than one value** at the same time.
- This is done by separating the return values with a comma.
- The returned values can be **unpacked** into multiple variables.

```
def calculate(a, b):
    return a+b, a-b, a*b
```

```
add, sub, mul = calculate(10, 5)
print("Addition:", add)
print("Subtraction:", sub)
print("Multiplication:", mul)
```

Output:

```
Addition: 15
Subtraction: 5
Multiplication: 50
```

Example 5: Recursive function

🔹 Theory:

- A **recursive function** is a function that calls itself.
- It is commonly used to solve problems that can be broken down into smaller, similar subproblems (e.g., factorial, Fibonacci series).
- Every recursive function needs a **base case** to stop the recursion.

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n-1)

print("Factorial of 5:", factorial(5))
```

Output:

```
Factorial of 5: 120
```

Summary: Functions are essential in Python because they reduce repetition, make programs modular, and increase readability.