Python Function Notes for Beginners

📘 1. What is a Function?

A **function** is a reusable block of code that performs a specific task.
Functions help make programs **modular**, **readable**, and **easy to maintain**.

✅ Example:

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

Output:

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

🧩 2. Syntax of a Function

```
def function_name(parameters):
    """Optional docstring"""
    # code block
    return value
```

✅ Example:

```
def add(a, b):
    return a + b


result = add(5, 3)
print(result)  # Output: 8
```

🧠 3. Types of Functions

1. **Built-in Functions** → Predefined in Python
   Examples: print(), len(), max(), sum(), type(), etc.
2. **User-defined Functions** → Created by the programmer
   Example:
3. def multiply(x, y):
4. return x * y
5. print(multiply(4, 5))

⚙️ **4. Function with and without Arguments**

(a) Without Arguments:

```
def say_hello():
    print("Hello World!")
say_hello()
```

(b) With Arguments:

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

🎯 **5. Function with Return Statement**

The return statement sends a value back to the caller.

```
def square(num):
    return num * num


print(square(4))  # Output: 16
```

If you don’t use return, the function returns None by default.

📦 6. Default Parameters

Default parameters are used when no value is passed for an argument.

```
def greet(name="Guest"):
    print("Hello", name)

greet()         # Output: Hello Guest
greet("Deepesh") # Output: Hello Deepesh
```

🔀 7. Keyword Arguments

You can specify arguments using their parameter names.

```
def student_info(name, age):
    print(f"Name: {name}, Age: {age}")

student_info(age=21, name="Alice")
```

📋 8. Arbitrary Arguments

(a) *args — Multiple Positional Arguments

Used when you don’t know how many arguments will be passed.

```
def add_numbers(*args):
    return sum(args)

print(add_numbers(2, 3, 5, 7))  # Output: 17
```

(b) **kwargs — Multiple Keyword Arguments

Used to accept many key-value pairs.

```
def display_info(**kwargs):
    for key, value in kwargs.items():
```

```
    print(key, ":", value)

display_info(name="John", age=25, city="Delhi")
```

🔁 9. Function Returning Multiple Values

You can return more than one value from a function using tuples.

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

add, sub, mul = operations(10, 5)
print(add, sub, mul)
```

Output:

15 5 50

🧱 10. Nested Functions (Function inside Function)

```
def outer():
    def inner():
        print("Inside inner function")
    print("Inside outer function")
    inner()

outer()
```

Output:

Inside outer function

Inside inner function

🔒 11. Lambda (Anonymous) Functions

A **lambda function** is a small, one-line function without a name.

Syntax:

lambda arguments: expression

Example:

```
square = lambda x: x * x

print(square(5))  # Output: 25
```

With multiple parameters:

```
add = lambda a, b: a + b

print(add(10, 20))  # Output: 30
```

🪆 12. Recursion (Function Calling Itself)

A function that calls itself is called **recursive**.
Commonly used in problems like factorial, Fibonacci, etc.

Example:

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


print(factorial(5))  # Output: 120
```

🧰 13. Built-in Functions (Common Examples)

Function	Description	Example
len()	Returns length	len("Hello") → 5

Function	Description	Example
max()	Returns max value	max([2, 5, 8]) → 8
min()	Returns min value	min([2, 5, 8]) → 2
sum()	Adds all items	sum([1, 2, 3]) → 6
type()	Returns data type	type(10) → <class 'int'>
sorted()	Returns sorted list	sorted([3,1,2]) → [1,2,3]
range()	Returns a sequence	range(5) → [0,1,2,3,4]

🧠 14. Scope and Lifetime of Variables

Type	Description
Local Variable	Declared inside a function; accessible only inside it.
Global Variable	Declared outside any function; accessible everywhere.

Example:

```
x = 10  # global variable


def show():
    y = 5  # local variable
    print("Inside function:", x + y)


show()
print("Outside function:", x)
```

🧩 15. Practical Example:

```
def student_report(name, marks):
```

```
    print(f"Student: {name}")
    for subject, score in marks.items():
        print(f"{subject}: {score}")

marks = {"Math": 85, "Science": 90, "English": 88}
student_report("Alice", marks)
```

Output:

Student: Alice

Math: 85

Science: 90

English: 88

💡 16. Key Points to Remember

- Use def to define a function.
- Use return to send a result back.
- Use *args and **kwargs for flexible arguments.
- Use lambda for small anonymous functions.
- Keep functions small and focused on one task.