Day25_Functions_Basics_and_Return_vs_Print

June 21, 2025

Day 25: Introduction to Functions in Python

In this notebook, I am documenting everything I learned about functions in Python — explained step-by-step in a simple way.

1 What is a Function?

A **function** is a reusable block of code that performs a specific task.

We define a function using the `def` keyword in Python.

Functions make our code more organized, reusable, and readable.

2 Types of Functions in Python

Python has two main types of functions:

2.1 Built-in Functions

- Already provided by Python
- We can use them directly without defining them
- Examples: `print()`, `len()`, `type()`, `int()`, `range()`, `sum()`, etc.

2.2 User-defined Functions

- Created by the user using `def`
- Helps organize and reuse custom logic

[1]:
```
# Example of Built-in Functions
print("Hello")           # Prints output
print(len("Python"))     # Returns length
print(type(5.2))         # Returns data type
print(sum([10, 20, 30])) # Sum of list
```

```
Hello
6
<class 'float'>
60
```

[2]:
```python
# Example of User-defined Function
def multiply(a, b):
    return a * b

print("Multiplication:", multiply(4, 5))
```

```
Multiplication: 20
```

2.3 When to Use Built-in vs User-defined

Built-in Function	Use when Python already provides the functionality
User-defined	Use when you need to create your own logic or reuse logic multiple times

3 Function Types (Based on Arguments & Return)

1. Function with **no arguments**, **no return**
2. Function with **arguments**, **no return**
3. Function with **arguments**, **and return**
4. Function inside a **class** (method)

3.1 Simple Example: greet function (no argument, no return)

[6]:
```python
# 1 Function with no arguments, no return

def greet():
    print("Hello! Welcome to Python.")

# Calling the function
greet()
```

```
Hello! Welcome to Python.
```

3.2 Function with two arguments (add)

[7]:
```python
# 2 Function with arguments, no return

def add(x, y):  # x and y are formal arguments
    c = x + y
    print("Sum is:", c)

# 5 and 6 are actual arguments passed to the function
add(5, 6)
```

```
Sum is: 11
```

3.3 Function with arguments and return

[ ]:
```python
# 3 Function with arguments and return

def multiply(x, y):
    return x * y

result = multiply(3, 4)
print("Multiplication:", result)
```

3.4 Function inside a class (method)

[5]:
```python
# 4 Function inside a class (method)

class Welcome:
    def greet(self):
        print("Hello from inside the class!")

# Create object
w = Welcome()
w.greet()
```

```
Hello from inside the class!
```

4 Return vs Print in Python Functions

In Python, both `print()` and `return` are used inside functions, but they serve **different purposes**.

- `print()` → just shows the output on the screen
- `return` → sends the value back to the caller (you can store it or use it later)

What is the Difference?

Feature	`print()`	`return`
Purpose	Shows output on the screen	Sends output back to the function caller
Used in	Small or demo programs	Big programs or reusable logic
Stores value?	No (just displays)	Yes (can assign to a variable)
Reusable?	No	Yes

4.1 When to Use `return`

Use `return` when you: - Want to store the output in a variable - Want to use the output in another function or calculation - Want the function to be reusable

[8]:
```python
def add(x, y):
    return x + y  # sends result back to caller

# Store the result
```

```
result = add(5, 3)
print("Returned value is:", result)

# Reuse the result
total = result * 2
print("Total after multiplying:", total)
```

```
Returned value is: 8
Total after multiplying: 16
```

4.2 When to Use `print`

Use `print()` when you: - Just want to **see the output** - Don't need to **store or reuse** the result - Are doing a **small task**, or testing

[9]:
```
def add(x, y):
    print("Sum is:", x + y)  # only shows output

add(5, 3)

# But you cannot store the result:
result = add(5, 3)  # This stores `None`
print("Result:", result)  # Output: None
```

```
Sum is: 8
Sum is: 8
Result: None
```

4.3 Summary

Use `print()` when…	Use `return` when…
You want to **just display** the result	You want to **store or reuse** the result
You're writing a **small demo or test**	You're building **reusable functions**
You don't need the result again	You plan to **use the result again**

In real-world projects, we **mostly use `return`** because we often need to pass results from one function to another.

5 Function with 1, 2, and 3 variables

5.1 Function with 1 variable

[ ]:
```
# 1 variable
def square(x):
    return x * x

print("Square:", square(4))
```

5.2 Function with 2 variable

[ ]:
```
# 2 variables
def subtract(x, y):
    return x - y

print("Subtraction:", subtract(10, 3))
```

5.3 Function with 3 variables

[ ]:
```
# 3 variables
def total(x, y, z):
    return x + y + z

print("Total of 3 numbers:", total(1, 2, 3))
```

5.4 What happens if we pass fewer arguments than required?

[11]:
```
# Function expects 3 arguments
def total(x, y, z):
    return x + y + z

# This will raise an error because only 2 values are passed

total(5, 6)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[11], line 7
      3     return x + y + z
      5 # This will raise an error because only 2 values are passed
----> 7 total(5, 6)

TypeError: total() missing 1 required positional argument: 'z'
```

6 Formal vs Actual Arguments

- **Formal Arguments** → The variables inside the function definition (e.g., x, y)
- **Actual Arguments** → The values you pass when calling the function (e.g., 5, 6)

[12]:
```
def add(x, y):  # x, y → formal
    return x + y

add(5, 6)  # 5, 6 → actual
```

[12]: 11

7 Recursive Function

A function that calls **itself** inside is called a recursive function.

[13]:
```
# Example: Print numbers from n to 1
def countdown(n):
    if n == 0:
        return
    print(n)
    countdown(n - 1)

countdown(5)
```

```
5
4
3
2
1
```

8 Function inside a Class = Method

When we define a function **inside a class**, it becomes a method.

[14]:
```
class Nit:
    def greet():  # No self → it's a static method
        print("Welcome from class method")

Nit.greet()
```

```
Welcome from class method
```

9 Summary

- **def** is used to define functions
- Functions can take **zero or many arguments**
- Use `return` when you want to **send data back**
- **Formal arguments**: in function definition
- **Actual arguments**: in function call
- Functions help avoid code repetition and improve readability
- You can also define **recursive functions** and **methods inside classes**