# Function in Python

Premanand S

Assistant Professor
School of Electronics Engineering
Vellore Institute of Technology *john@smith.com*

October 23, 2024

## What is a Function?

- A **function** is a block of reusable code that performs a specific task.
- It allows you to organize and reuse code effectively.
- A function can take **inputs** (parameters or arguments) and can return an **output**.

## Why are Functions Used in Programming?

- Functions help organize code into smaller, manageable pieces.
- You can avoid repeating code by reusing functions.
- Functions make programs easier to debug and maintain.

## Benefits of Using Functions

- **Modularity:** Break down complex tasks into smaller, independent units.
- **Code Reusability:** Reuse functions in different parts of the program or in other programs.
- **Maintainability:** Update functions without affecting other parts of the program.
- **Readability:** Improves the structure and clarity of the code.
- **Abstraction:** Hide implementation details and simplify function usage.

## Basic Syntax of a Python Function

- `def`: Keyword to define a function.
- `function_name`: Descriptive name of the function.
- `parameters`: (Optional) Inputs to the function.
- `return`: (Optional) The result the function returns.

**Example (Python Code)**

```
def function_name(parameters):
    # Function body
    # Code block to perform the task
    return result  # Optional
```

## Example of a Simple Function

### Example (Python Code)

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

# Calling the function
greet()

Output:

Hello, welcome to Python functions!
```

## How to Define a Function

- To define a function in Python, use the `def` keyword, followed by the function name, parentheses `()` (which can include parameters), and a colon `:`. The function body is indented under the definition.

**Example (Python Code)**

```
def function_name(parameters):
    # Function body
    # Code block to perform the task
```

## Example of Defining and Calling a Function

- Here's a simple example of defining and calling a function:

**Example (Python Code)**

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

#To call the function, simply write:
greet()
```

## The Importance of Indentation

- In Python, indentation is crucial for defining the structure of the code.
- The code block within a function must be indented consistently.
- Indentation defines the scope of the function.
- All code within the function must be indented to be recognized as part of that function.
- Incorrect indentation will lead to **IndentationError** or unexpected behavior.

### Example (Python Code)

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

## Understanding Parameters and Arguments

- **Parameters** are the variables listed in a function's definition.
- **Arguments** are the values that are passed to the function when it is called.
- Functions can take different types of parameters, which affect how arguments can be passed.

## Types of Parameters

- Positional Arguments: Passed based on their position in the function call.
- Keyword Arguments: Passed by explicitly stating the parameter name and its value.
- Default Parameters: Parameters that have a default value if no argument is provided.
- Arbitrary Arguments: Allow passing a variable number of arguments (e.g., using ‘*args‘ and ‘**kwargs‘).

## Positional Arguments

- Arguments can be passed to functions by position. The order in which the arguments are passed to the function matters, as they are assigned to the corresponding parameters in the same order

**Example (Python Code)**

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

# Calling the function with positional arguments
result = add(3, 5)  # 3 is assigned to a, and 5 to b

The output will be:
result = 8
```

## Keyword Arguments

- Keyword arguments allow you to pass arguments to a function using the names of the parameters.
- This enables you to specify which parameter corresponds to which value, regardless of the order.

### Example (Python Code)

```
def describe_pet(pet_name, animal_type='dog'):
    print(f"I have a {animal_type} named {pet_name}.")

# Calling the function with keyword arguments
describe_pet(pet_name='Buddy', animal_type='cat')

The output will be:
I have a cat named Buddy.
```

## Parameters with Default Values

- Parameters can be assigned default values. If an argument for that parameter is not provided when the function is called, the default value is used.

### Example (Python Code)

```
def greet(name, greeting='Hello'):
    print(f"{greeting}, {name}!")

# Calling the function with default parameter
greet('Python')  # Uses the default greeting

The output will be:
Hello, Python
You can also specify the greeting:
greet('Prem', greeting='Hi')
Hi, Prem!
```

## Arbitrary Arguments

- Use ‘*args‘ to pass a variable number of positional arguments.
- Use ‘**kwargs‘ to pass a variable number of keyword arguments

### Example (Python Code)

```
def collect_args(*args):
    return args
print(collect_args(1, 2, 3))

def collect_kwargs(**kwargs):
    return kwargs
print(collect_kwargs(name='anand', age=38))
```