# **Function in Python Part 1**

### Create FUnction | Types of function | Calling | Types of Arguments

**1. Introduction to Functions**

**Definition of functions:**

Functions in Python are named blocks of code that perform a specific task. They allow for code organization and reusability by encapsulating a set of instructions under a single name.

**Purpose and benefits:**

* Functions serve the purpose of breaking down complex tasks into smaller, manageable chunks.
* They promote code readability, modularity, and abstraction, making it easier to understand and maintain codebases.
* Additionally, functions facilitate code reuse, reducing redundancy and promoting efficiency.

**Reusability and modularity:**

* Functions can be reused multiple times within a program or across different programs. This reusability promotes modular programming, where complex systems are built from smaller, self-contained modules (functions).

**2. Types of Functions**

I. **Built-in functions:**

These are predefined functions provided by Python, **such as `print()`, `len()`, `max()`, etc.**

**Note:** They are readily available for use without the need for additional implementation.

```

**# Built-in function example**

print(len("Hello")) # Output: 5

```

Il. **User-defined functions:**

These are functions created by the user to fulfill specific requirements. Users can define their own functions using the `**def` keyword** and customize them according to their needs.

**# Example of a User-defined function**

```

def greet():

print("Hello, welcome to the world of functions!")

# Calling the function

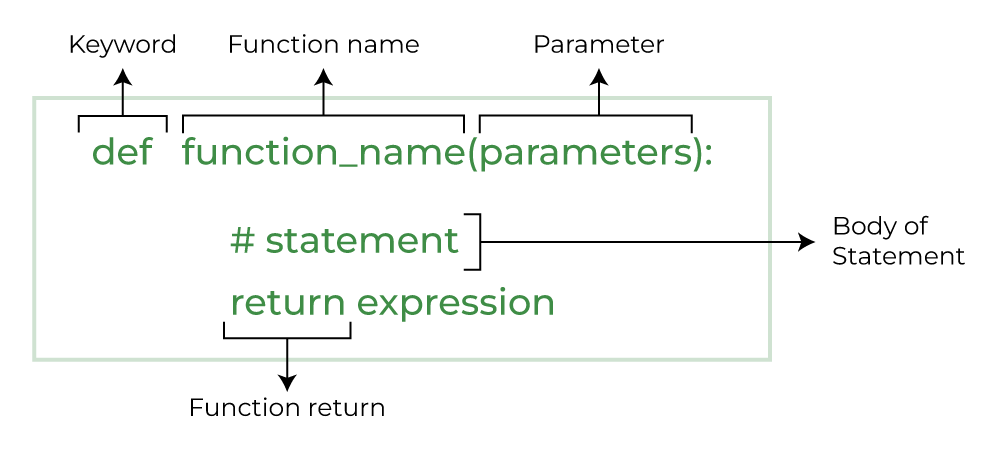
greet()

```

**3. Creating Functions**

**Syntax of function declaration:**

Functions are declared using the `**def` keyword** followed by the function name and parentheses containing any parameters. The function block is then defined with an indented code block.



**Defining function blocks**:

Inside the function block, the set of instructions or code statements to be executed are defined. This block can contain any valid Python code, including variable declarations, conditional statements, loops, etc.

**Using the `def` keyword:** The `def` keyword is used to define functions in Python. It signifies the beginning of a function definition and is followed by the function name and parameters.

**4. Calling Functions**

**Invoking functions:**

Functions are invoked or called by using their name followed by parentheses containing any arguments (if required). This execution triggers the function to perform its defined task.

**Passing arguments:**

Arguments can be passed to functions within the parentheses during function invocation. These arguments provide input values to the function for processing.

**Return values:**

Functions can optionally return a value or result using the `return` statement. This allows the function to communicate information back to the caller. If no `return` statement is specified, the function returns `None` by default.

**```**

**# Invoking functions**

def greet():

print("Hello!")

greet()

**# Passing arguments**

def greet(name):

print("Hello,", name)

greet("Alice")

**# Return values**

def add(a, b):

return a + b

result = add(3, 4)

print(result) # Output: 7

**```**

**5. Function Parameters**

**Positional parameters:**

These are parameters that are passed to a function based on their position or order in the function signature. The values are assigned to the parameters in the order they are provided.

**Default parameters:**

Default parameters have predefined values that are used when no argument is provided during function invocation. They offer flexibility by allowing functions to be called with fewer arguments.

**Keyword parameters:**

Keyword parameters are passed to a function with explicit parameter names, allowing for greater clarity and flexibility in function calls. The order of keyword arguments does not matter.

**Arbitrary arguments (`\*args`, `\*\*kwargs`):**

These special parameters allow functions to accept a variable number of arguments. `\*args` is used to pass a variable number of positional arguments, while `\*\*kwargs` is used to pass a variable number of keyword arguments. They provide flexibility when the number of arguments is uncertain or varies.

```

**# Positional parameters**

def greet(name, age):

print("Hello,", name, "you are", age, "years old.")

greet("Alice", 25)

**# Default parameters**

def greet(name="Guest"):

print("Hello,", name)

greet() # Output: Hello, Guest

**# Keyword parameters**

def greet(name, message):

print(message, name)

greet(message="Welcome", name="Alice")

**# Arbitrary arguments (\*args)**

def add(\*args):

total = 0

for num in args:

total += num

return total

print(add(1, 2, 3, 4)) # Output: 10

**# Arbitrary keyword arguments (\*\*kwargs)**

def person\_info(\*\*kwargs):

for key, value in kwargs.items():

print(key + ":", value)

person\_info(name="Alice", age=25, country="USA")

```