**Function:**

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

**Syntax:**

def function\_name(parameters):

# code to be executed

# ...

return result # optional

**def:**

This keyword is used to define a function.

**function\_name** This is the name of the function. Choose a descriptive name that indicates what the function does.

**parameters**: These are the input values that the function accepts. Parameters are optional.: **colon:** It signifies the beginning of the function body.

**Function body:** This is the indented block of code that defines what the function does.

**return:** This keyword is used to specify the value that the function should return. The return statement is optional.

**Example:1**

##USING PRINT##

def add(x,y):

    print(x+y)

add(5,7)

**Example:2**

##USING RETURN##

def add(a,b):

    return a+b

x=int(input("Enter the Number1:"))

y=int(input("Enter the Number2:"))

print(add(x,y))

**Application:**

**Code Organization:**

* + Functions help in structuring code, making it more organized and modular. Code can be broken down into smaller, manageable pieces, improving readability and maintainability.

1. **Code Reusability:**
   * Once a function is defined, it can be reused in different parts of the program or even in other programs. This promotes the Don't Repeat Yourself (DRY) principle.
2. **Abstraction:**
   * Functions allow you to abstract away the implementation details of a particular task. Users of the function only need to know what the function does, not how it accomplishes it.
3. **Parameterization:**
   * Functions can accept parameters, allowing them to work with different input values. This makes functions flexible and adaptable to various scenarios.
4. **Return Values:**
   * Functions can return values, providing a way to get results or feedback from a specific operation. This is crucial for building versatile and interactive programs.
5. **Modularity:**
   * Functions enable the development of modular code, where each function serves a specific purpose. This modularity simplifies debugging, testing, and maintenance.
6. **Event Handling:**
   * In graphical user interface (GUI) programming, functions often handle events like button clicks or keypresses. Each event is associated with a specific function to execute when triggered.
7. **Data Analysis and Manipulation:**
   * Functions are extensively used in data analysis and manipulation tasks. Libraries like NumPy and pandas provide functions for working with arrays and data frames, respectively.
8. **Mathematical Operations:**
   * Functions are fundamental for performing mathematical operations. Python's standard library and external libraries offer a wide range of mathematical functions for tasks like trigonometry, logarithms, etc.
9. **File Operations:**
   * Functions are employed for reading from and writing to files. File I/O functions help in managing data persistence.
10. **Web Development:**
    * In web development, functions are used to define the behavior of different parts of a web application. Frameworks like Flask and Django heavily rely on functions to handle HTTP requests and generate responses.
11. **Testing:**
    * Functions make it easier to write unit tests for different parts of your code. This is essential for ensuring that each function behaves as expected.

**How to write a user defined functions.**

### **1. **Function Definition:****

Start by using the def keyword to define your function. Choose a meaningful name for your function that reflects its purpose. Use parentheses () to enclose any parameters the function may take.

def my\_function(parameter1, parameter2):

# Function body goes here

# ...

# You can use the parameters in the function code

### 2. ****Function Body:****

Indent the code following the colon (:) to indicate the function body. This is the block of code that gets executed when the function is called.

def my\_function(parameter1, parameter2):

# Function body

result = parameter1 + parameter2

print("Result:", result)

### 3. ****Return Statement (Optional):****

If your function needs to return a value, use the return statement. This is optional; if omitted, the function returns None.

def add\_numbers(x, y):

result = x + y

return result

### 4. ****Function Call:****

To use your function, call it with the required arguments (if any). Assign the result to a variable if the function returns a value.

result = add\_numbers(3, 5)

print(result)

Example:Assignment

**Different Type of Function Argument**

### **1.Default Arguments:**

Default arguments are used to provide a default value for a parameter if a value is not provided during the function call.

**Example:1**

def power(base, exponent=2):

return base \*\* exponent

# Calling the function with the default exponent (2)

result1 = power(2)

# Calling the function with a specified exponent (3)

result2 = power(2, 3)

# Printing the results

print("Result 1:", result1) # Output: Result 1: 4 (2^2)

print("Result 2:", result2) # Output: Result 2: 8 (2^3)

### **2.Keyword Arguments:**

Keyword arguments allow you to pass values to a function using the parameter names. This enables you to pass values in any order and makes the function call more self-explanatory.

def greet(first\_name, last\_name):

print('First Name:', first\_name)

print('Last Name:', last\_name)

# Calling the function with keyword arguments

greet(last\_name='Cartman', first\_name='Eric')

### **3.Required Arguments:**

Required arguments are parameters that must be passed during the function call. If not provided, the function will raise an error.

Example:

def greet(name, greeting):

"""This function greets a person with a specific greeting."""

print(f"{greeting}, {name}!")

# Calling the function with required arguments

greet("Alice", "Hello")

### **4.Variable-Length Arguments:**

Python allows variable-length arguments for both positional and keyword arguments.

**Example with \*args**

def myFun(\*argv):

for arg in argv:

print(arg)

# Calling the function with multiple arguments

myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks')

The \*args syntax allows the function to accept any number of positional arguments, which are collected into a tuple.

**Example with \*kwargs**

The **\*\*kwargs** syntax allows the function to accept any number of keyword arguments, which are collected into a dictionary.

Example:

def myFun(\*\*kwargs):

d = {}

for key, value in kwargs.items():

d[key] = value

print(d)

# Calling the function with multiple keyword arguments

myFun(first='Geeks', mid='for', last='Geeks')