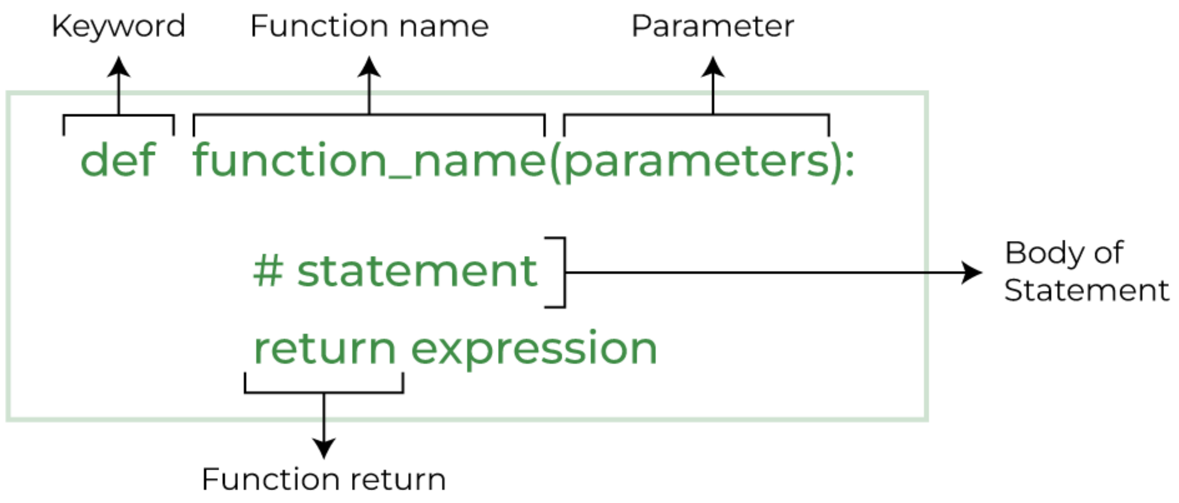
Python Functions

Python Functions are a block of statements that does a specific task. The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Function Declaration

The syntax to declare a function is:



Syntax of Python Function Declaration

Defining a Function

We can define a function in Python, using the **def keyword**. We can add any type of functionalities and properties to it as we require.

The def keyword stands for define. It is used to create a **user-defined function**. It marks the beginning of a function block and allows you to group a set of statements so they can be reused when the function is called.

Syntax:

def function_name(parameters):
# function body

Explanation:

- **def:** Starts the function definition.

- **function_name:** Name of the function.
- **parameters:** Inputs passed to the function (inside ()), optional.
- **Indented code:** The function body that runs when called.

Example: Let’s understand this with a simple example. Here, we define a function using def that prints a welcome message when called.

```
def fun():
  print("Welcome to GFG")
```

Calling a Function

After creating a function in Python we can call it by using the name of the functions followed by parenthesis containing parameters of that particular function. Below is the example for calling def function Python.

```
def fun():
  print("Welcome to GFG")


fun() # Driver code to call a function
```

Output

```
Welcome to GFG
```

Function Arguments

Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.

Syntax:

def function_name(parameters):
"""Docstring"""
# body of the function
return expression

Let's understand this with an example, we will create a simple function in Python to check whether the number passed as an argument to the function is even or odd.

```
def evenOdd(x):
    if (x % 2 == 0):
        return "Even"
    else:
        return "Odd"


print(evenOdd(16))
print(evenOdd(7))
```

Output

```
Even
Odd
```

Types of Function Arguments

Python supports various types of arguments that can be passed at the time of the function call. In Python, we have the following function argument types in Python, Let's explore them one by one.

1. Default Arguments

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments to write functions in Python.

```
def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)


myFun(10)
```

Output

```
x:  10
y:  50
```

2. Keyword Arguments

In keyword arguments, values are passed by explicitly specifying the parameter names, so the order doesn't matter.

```
def student(fname, lname):
    print(fname, lname)


student(fname='Geeks', lname='Practice')
student(lname='Practice', fname='Geeks')
```

Output

```
Geeks Practice
Geeks Practice
```

3. Positional Arguments

In positional arguments, values are assigned to parameters based on their order in the function call.

```
def nameAge(name, age):
    print("Hi, I am", name)
    print("My age is ", age)


print("Case-1:")
nameAge("Suraj", 27)


print("\nCase-2:")
nameAge(27, "Suraj")
```

Output

```
Case-1:
```

```
Hi, I am Suraj

My age is  27
```

Case-2:

```
Hi, I am 27

My age is  Suraj
```

4. Arbitrary Arguments

In Python Arbitrary Keyword Arguments, *args and **kwargs can pass a variable number of arguments to a function using special symbols. There are two special symbols:

- ***args** in Python (Non-Keyword Arguments)
- ****kwargs** in Python (Keyword Arguments)

Example: Here's an example that separately shows non-keyword (*args) and keyword (**kwargs) arguments in the same function.

```
def myFun(*args, **kwargs):

    print("Non-Keyword Arguments (*args):")

    for arg in args:

        print(arg)


    print("\nKeyword Arguments (**kwargs):")

    for key, value in kwargs.items():

        print(f"{key} == {value}")


# Function call with both types of arguments

myFun('Hey', 'Welcome', first='Geeks', mid='for', last='Geeks')
```

Output

```
Non-Keyword Arguments (*args):
```

```
Hey
Welcome


Keyword Arguments (**kwargs):
first == Geeks
mid == for
last == Geeks
```

Function within Functions

A function defined inside another function is called an inner function (or nested function). It can access variables from the enclosing function’s scope and is often used to keep logic protected and organized.

```
def f1():
    s = 'I love GeeksforGeeks'
    def f2():
        print(s)


    f2()
f1()
```

Output

```
I love GeeksforGeeks
```

Anonymous Functions

In Python, an anonymous function means that a function is without a name. As we already know the def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions.

```
def cube(x): return x*x*x   # without lambda

cube_l = lambda x : x*x*x  # with lambda
```

```
print(cube(7))
print(cube_l(7))
```

Output

```
343
343
```

Return Statement in Function

The return statement ends a function and sends a value back to the caller. It can return any data type, multiple values (packed into a tuple), or None if no value is given.

Syntax:

return [expression]

Parameters: return ends the function, **[expression]** is the optional value to return (defaults to None)**.**

```
def square_value(num):
    """This function returns the square
    value of the entered number"""
    return num**2


print(square_value(2))
print(square_value(-4))
```

Output

```
4
16
```

Pass by Reference and Pass by Value

In Python, variables are references to objects. When we pass them to a function, the behavior depends on whether the object is mutable (like lists, dictionaries) or immutable (like integers, strings, tuples).

- **Mutable objects:** Changes inside the function affect the original object.
- **Immutable objects:** The original value remains unchanged.

```
# Function modifies the first element of list
def myFun(x):
    x[0] = 20


lst = [10, 11, 12, 13]
myFun(lst)
print(lst)   # list is modified


# Function tries to modify an integer
def myFun2(x):
    x = 20


a = 10
myFun2(a)
print(a)     # integer is not modified
```

Output

```
[20, 11, 12, 13]
10
```

Note: *Technically, Python uses "pass-by-object-reference". Mutable objects behave like pass by reference, while immutable objects behave like pass by value*

Recursive Functions

A recursive function is a function that calls itself to solve a problem. It is commonly used in mathematical and divide-and-conquer problems. Always include a base case to avoid infinite recursion.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)


print(factorial(4))
```

Output

```
24
```

Here we have created a recursive function to calculate the factorial of the number. It calls itself until a base case (n==0) is met.