

Functions

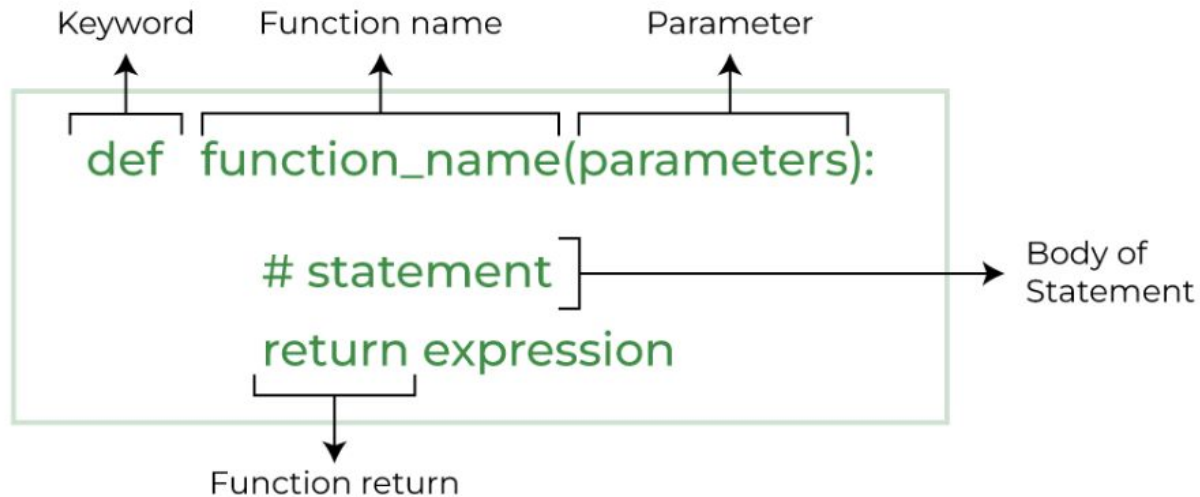
Functions

- **Python Functions** is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks 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.
- Some **Benefits of Using Functions**
- Increase Code Readability
- Increase Code Reusability

Function Declaration

Python Function Declaration

The syntax to declare a function is:



Syntax of Python Function Declaration

Types of Functions in Python

- Below are the different types of functions in [Python](#):
- **Built-in library function:** These are [Standard functions](#) in Python that are available to use.
- **User-defined function:** We can create our own functions based on our requirements.

Creating a Function in Python

- We can define a function in Python, using the **def** keyword. Then call it by using the name

Python



```
1  # A simple Python function
2  def fun():
3      print("Welcome to GFG")
4
5
6  # Driver code to call a function
7  fun()
```

Python Function with Parameters

Python Function Syntax with Parameters

```
def function_name(parameter: data_type) -> return_type:  
    """Docstring"""  
    # body of the function  
    return expression
```

Python



```
1 def add(num1: int, num2: int) -> int:
2     """Add two numbers"""
3     num3 = num1 + num2
4
5     return num3
6
7 # Driver code
8 num1, num2 = 5, 15
9 ans = add(num1, num2)
10 print(f"The addition of {num1} and {num2} results
    {ans}.")
```

Output:

The addition of 5 and 15 results 20.



```
1  # some more functions
2  def is_prime(n):
3      if n in [2, 3]:
4          return True
5      if (n == 1) or (n % 2 == 0):
6          return False
7      r = 3
8      while r * r <= n:
9          if n % r == 0:
10             return False
11             r += 2
12         return True
13 print(is_prime(78), is_prime(79))
```

Output:

False True

Types of Python Function Arguments

- **Default argument**
- **Keyword arguments (named arguments)**
- **Positional arguments**
- **Arbitrary arguments** (variable-length arguments `*args` and `**kwargs`)

Python Function Arguments

```
1  # A simple Python function to check
2  # whether x is even or odd
3  def evenOdd(x):
4      if (x % 2 == 0):
5          print("even")
6      else:
7          print("odd")
8
9
10 # Driver code to call the function
11 evenOdd(2)
12 evenOdd(3)
```

Default argument



```
1  # Python program to demonstrate
2  # default arguments
3  def myFun(x, y=50):
4      print("x: ", x)
5      print("y: ", y)
6
7
8  # Driver code (We call myFun() with only
9  # argument)
10 myFun(10)
```

Output:

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

Keyword Argument



```
1  # Python program to demonstrate Keyword Arguments
2  def student(firstname, lastname):
3      print(firstname, lastname)
4
5
6  # Keyword arguments
7  student(firstname='Geeks', lastname='Practice')
8  student(lastname='Practice', firstname='Geeks')
```

Output:

```
Geeks Practice
Geeks Practice
```

Positional Argument





```
1 def nameAge(name, age):
2     print("Hi, I am", name)
3     print("My age is ", age)
4
5
6 # You will get correct output because
7 # argument is given in order
8 print("Case-1:")
9 nameAge("Suraj", 27)
10 # You will get incorrect output because
11 # argument is not in order
12 print("\nCase-2:")
13 nameAge(27, "Suraj")
```

Arbitrary Keyword 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)



```
1  # Python program to illustrate
2  # *args for variable number of arguments
3  def myFun(*argv):
4      for arg in argv:
5          print(arg)
6
7
8  myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks')
```



```
1 # Python program to illustrate
2 # *kwargs for variable number of keyword arguments
3
4
5 def myFun(**kwargs):
6     for key, value in kwargs.items():
7         print("%s == %s" % (key, value))
8
9
10 # Driver code
11 myFun(first='Geeks', mid='for', last='Geeks')
```

Output:

```
first == Geeks
mid == for
last == Geeks
```

Recursive functions



```
1 def factorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * factorial(n - 1)
6
7 print(factorial(4))
```


Returning Multiple Values from a Function

```
python

def square_and_cube(num):
    return num ** 2, num ** 3

# Unpack the returned values
square, cube = square_and_cube(4)
print("Square:", square)
print("Cube:", cube)
```

Using Lambda Functions with map

python

```
# Using lambda function with map to square each element in a list
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x**2, numbers))
print(squared_numbers)
```

Using Lambda Functions with filter

Combining Lambda Functions with reduce (from functools):

python

```
# Using lambda function with filter to keep only even numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)
```

python

```
from functools import reduce

# Using lambda function with reduce to calculate the sum of a list
numbers = [1, 2, 3, 4, 5]
sum_of_numbers = reduce(lambda x, y: x + y, numbers)
print(sum_of_numbers)
```

Nested functions

Nested functions in Python refer to functions defined inside another function.

```
python

def outer_function(x):
    def inner_function(y):
        return y * 2

    def another_inner(z):
        return z ** 2

    result = inner_function(x) + another_inner(x)
    return result

output = outer_function(5)
print(output)
```

LS Algorithm

Our goal is to estimate $(\hat{A}, \hat{X}) = \min_{A, X} \frac{1}{2} \|Y - AX\|_F^2$, s.t. $A \geq 0, X \geq 0$.

However, it is very difficult to solve such a multivariate problem.

The ALS strategy is to solve following sub-problems alternately.

sub-problem for A

$$\min \frac{1}{2} \|Y - AX\|_F^2 \quad (8)$$

$$\text{s.t. } A \geq 0 \quad (9)$$

sub-problem for X

$$\min \frac{1}{2} \|Y - AX\|_F^2 \quad (10)$$

$$\text{s.t. } X \geq 0 \quad (11)$$

Solution of Least Squares

Here, I show basic calculation of least squares. First, objective function can be transformed as

$$\frac{1}{2} \|Y - AX\|_F^2 \quad (12)$$

$$= \frac{1}{2} \text{tr}(Y - AX)(Y - AX)^T \quad (13)$$

$$= \frac{1}{2} \text{tr}(YY^T - 2AXY^T + AXX^T A^T) \quad (14)$$

The stationary points \hat{A} can be found by equating the gradient components to 0:

$$\frac{\partial}{\partial A^T} \frac{1}{2} \text{tr}(YY^T - 2AXY^T + AXX^T A^T) \quad (15)$$

$$= -XY^T + XX^T A^T = 0 \quad (16)$$

Therefore,

$$\hat{A}^T = (XX^T)^{-1}XY^T. \quad (17)$$

