Example:
```
def max2(n1,n2):
   if n1>n2:
      print(f'{n1} is max')
   else:
      print(f'{n2} is max')

max2(10,2)
max2(10,100)
max2(200,90) # Positional
max2(n1=5,n2=6) # Keyword
```

Output
```
10 is max
100 is max
200 is max
6 is max
```

Function with positional only required arguments or parameters

Function with positional only required parameters send values using parameter position but name.

Syntax:
```
def  function-name(param,param,param,/):
       statement-1
        statement-2
```

/ indicates given parameter list is positional only.

Example:
```
def count_vowels(string,/):
   c=0
   for ch in string:
      if ch in "aeiouAEIOU":
         c=c+1
   print(f'Vowel Count is {c}')
```

```
count_vowels("python")
count_vowels("oracle")
#count_vowels(string="java")
```

Output

```
Vowel Count is 1
Vowel Count is 3
```

Function with required keyword only parameters or arguments

Function with required keyword only parameter receives value using parameter names but not using positions.

Syntax:

```
def function-name(*,param,param,param,..):
        statement-1
        statement-2
```

*indicates the parameter list is keyword only

Example:

```
def count_special(*,string):
    c=0
    for ch in string:
        if not ch.isalnum():
            c=c+1
    print(f'Count of Special Characters {c}')


count_special(string="nit$23@")
#count_special("nit")
```

Output

```
Count of Special Characters 2
```

Example:
```
def fun1(a,b,/,*,c,d):
   print(a,b,c,d)



fun1(10,20,c=50,d=60)
```

Output
```
10 20 50 60
```

return

"return" is a keyword used inside function. It is branching statement in python. This statement is used to return value to caller or calling function.

return keyword after returning value it terminates execution of function.

return keyword returns only one object.

Syntax: return [value]

Whenever function returns value, it is assigned to variable.

Example:
```
def power(num,p):
   r=num**p
   return r


res=power(5,2)
print(res)
```

Output
```
25
```

Example:
```
def fun1():
    return 10
    return 20
    return 30


x=fun1()
print(x)
```

Output
```
10
```

Example:
```
def fun1():
    return 10,20,30,40,50


x=fun1()
print(x)
```

Output
```
(10, 20, 30, 40, 50)
```

Example:
```
def uppercase(string):
    output=''
    for ch in string:
        if ch>='a' and ch<='z':
            output=output+chr(ord(ch)-32)
        else:
            output=output+ch
    return output


str1=uppercase("abc")
print(str1)
```

Output

```
ABC
```

Example:

```
def isprime(num):
    c=0
    for i in range(1,num+1):
        if num%i==0:
            c=c+1
        if c>2:
            break
    return c==2


number=int(input("Enter any number "))
if isprime(number):
    print("Prime")
else:
    print("Not Prime")
```

Output

```
Enter any number 5
Prime

Enter any number 8
Not Prime
```

Function with optional arguments or default arguments or parameters

Default parameters are given value at the time defining function. This parameter not required value at the time of invoking function or executing function.

Syntax:

```
def function-name(param-name,param-name,param-
name=value,param-name=value,..):
        statement-1
        statement-2
```

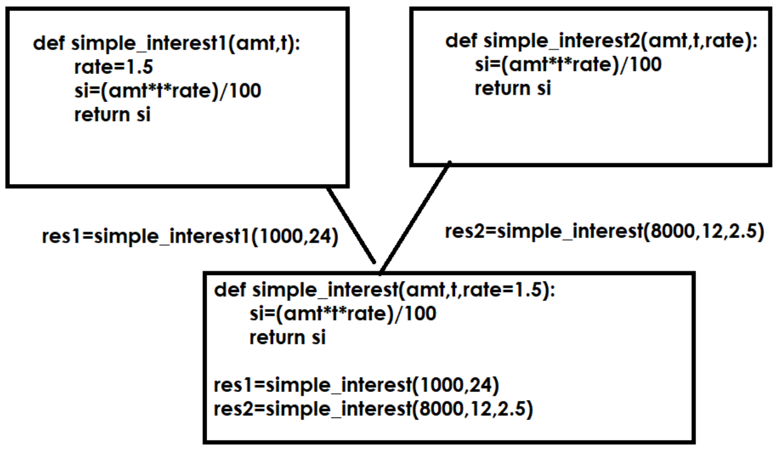
Example:

```
def fun1(a,b=0):
   print(a,b)

fun1(10)
fun1(100,200)
```

Output

```
10 0
100 200
```



```
def simple_interest1(amt,t):
      rate=1.5
      si=(amt*t*rate)/100
      return si
```

```
def simple_interest2(amt,t,rate):
     si=(amt*t*rate)/100
     return si
```

```
res1=simple_interest1(1000,24)
```

```
res2=simple_interest(8000,12,2.5)
```

```
def simple_interest(amt,t,rate=1.5):
     si=(amt*t*rate)/100
     return si

res1=simple_interest(1000,24)
res2=simple_interest(8000,12,2.5)
```

Example:

```
def sort(sequence,*,reverse=False):
   sorted_sequence=list(sequence)
   if reverse:
      for i in range(len(sequence)):
         for j in range(len(sequence)-1):
            if sorted_sequence[j]<sorted_sequence[j+1]:
```

```
sorted_sequence[j],sorted_sequence[j+1]=sorted_sequence[j+1],sort
ed_sequence[j]
    else:
       for i in range(len(sequence)):
           for j in range(len(sequence)-1):
               if sorted_sequence[j]>sorted_sequence[j+1]:

sorted_sequence[j],sorted_sequence[j+1]=sorted_sequence[j+1],sort
ed_sequence[j]

    return tuple(sorted_sequence)


A=[5,8,1,3,6,4]
B=sort(A)
print(A)
print(B)
C=sort(A,reverse=True)
print(C)
```

Output

```
[5, 8, 1, 3, 6, 4]
(1, 3, 4, 5, 6, 8)
(8, 6, 5, 4, 3, 1)
```

Example:

```
def draw_line(ch='-',size=20):
    for i in range(size):
        print(ch,end='')
    print()


draw_line()
draw_line(size=10)
draw_line(ch='*')
draw_line(ch='$',size=30)
```

Output

```
--------------------
----------
********************
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Function with variable length arguments or parameters

Function with variable length arguments or parameters receive 0 or more values (one parameter assigns multiple values).

Variable length arguments or parameters are two types

1. Variable length positional parameters or arguments
2. Variable length keyword parameters or arguments

Variable length positional parameters
Function with variable length positional parameters receives 0 or more values and store inside tuple.
Variable length positional parameter is prefix with *
Variable length positional parameter is of type tuple
A function is defined with one variable length positional parameter

Syntax:

```
def function-name(*param-name):
     statement-1
     statement-2
```

Example

```
def fun1(*a):
   print(a,type(a))

def fun2(*a):
   print(a)

fun1()
fun1(10)
fun1(10,20,30)
```

```
fun1(10,"naresh",1000.0)
fun2(10,20,30)
```

Output

```
() <class 'tuple'>
(10,) <class 'tuple'>
(10, 20, 30) <class 'tuple'>
(10, 'naresh', 1000.0) <class 'tuple'>
(10, 20, 30)
```