Function with required arguments or parameters
Function with required parameters required arguments/values at the time of invoking function or calling the function.

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

Example:
```
def fun1(a,b): # required parameter
    print(a,b)


fun1(100,200) # required positional arguments
fun1(b=300,a=400) # required keyword arguments
fun1(1.5,2.5)
fun1("python",3.13)
```

Output
```
100 200
400 300
1.5 2.5
python 3.13
```

if the values are given to function with parameter positions, it is called positional arguments
if the values are given to function with parameter names, it is called keyword arguments

```
def simple_interest(amt,t,rate):
    si=(amt*t*rate)/100
    print(f'Simple Interest {si:.2f}')



simple_interest(5000,12,1.5) #Positional Arguments
simple_interest(rate=2.0,amt=9000,t=24) #Keyword arguments
```

Output

Simple Interest 900.00
Simple Interest 4320.00

Defining function with required positional only arguments
This function accepts values of parameters using position only, it does not allow with parameter name.

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

Example:
```
def add(a,b,/): # Postional only
   print(f'Sum of {a} and {b} is {a+b}')
def sub(a,b,/): # Positional only
   print(f'Diff of {a} and {b} is {a-b}')

add(10,20)
#add(b=100,a=200)
sub(10,5)
#sub(b=10,a=5)
```

Output
Sum of 10 and 20 is 30
Diff of 10 and 5 is 5
Diff of 5 and 10 is -5

Defining function with required keyword only arguments
If function is defined required keyword only arguments, the value must be given to parameters with parameter names but not with positions.

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

Example:
```
def maximum(*,a,b):
```

```
    if a>b:
        print(f'{a} is max')
    else:
        print(f'{b} is max')



maximum(a=10,b=20)
maximum(b=100,a=200)
#maximum(10,20)
```

Output

```
20 is max
200 is max
```

Example:

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


fun1(10,20,c=30,d=40)
fun1(100,200,d=400,c=300)
```

Output

```
10 20 30 40
100 200 300 400
```

return keyword

A function returns values/result to calling function or caller using return keyword. “**return**” is called branching statement or passes control statement, this statement passes the execution control from called function to calling function or place.

“return” keyword after returning value terminates execution of function.

A function returns only one object or value. In order to return multiple values, it has to be grouped within collection type.

Syntax: return [value/expression]

Example:

```
def maximum(a,b):
    if a>b:
        return a
    else:
        return b



c=maximum(10,20)
print(c)
d=maximum(200,100)
print(d)
```

Output

```
20
200
```

Example:

```
def fun1():
    return 100
    return 200
    return 300

def fun2():
    return 100,200,300
def fun3():
    print("Hello")
    return
    print("Bye")

x=fun1()
print(x)
y=fun2()
print(y)
z=fun3()
print(z)
```

Output

```
100
(100, 200, 300)
Hello
None
```

Example:

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

def iseven(num):
    return num%2==0

def ispalnum(num):
    num1=num
    rev=0
    while num!=0:
        d=num%10
        rev=(rev*10)+d
        num=num//10
    return num1==rev

def count_digits(num):
    c=0
    while num>0:
        c=c+1
        num=num//10
    return c

def sum_digits(num):
    s=0
    while num>0:
        d=num%10
        s=s+d
        num=num//10
    return s
```

```
res1=isprime(7)
res2=sum_digits(123)
res3=count_digits(4567)
res4=ispalnum(121)
print(res1,res2,res3,res4)
res5=iseven(5)
print(res5)
```

Output

```
True 6 4 True
False
```

Example:

```
def vowel_count(s):
    c=0
    for ch in s:
        if ch in "aeiouAEIOU":
            c=c+1
    return c

def ispal(s):
    return s==s[::-1]


res1=vowel_count("java")
res2=ispal("madam")
print(res1,res2)
```

Output

```
2 True
```

Function with default parameters or optional parameters

Default parameters are given values at the time of defining function or writing function.

If value of these parameters is not given at the time invoking function or calling function, it assigns default values.

Syntax:

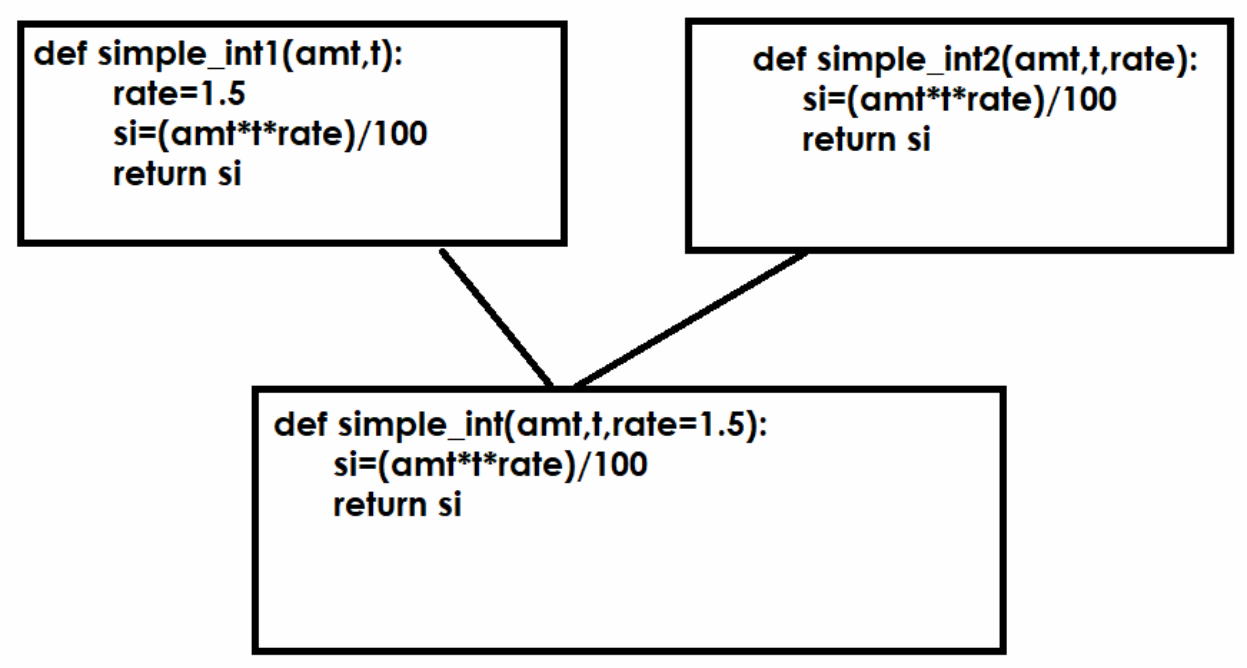
```
def function-name(param-name,param-name=value,param-name=value,..):
        statement-1
        statement-2
```

Example:

```
def fun1(a,b,c):
    print(a,b,c)

def fun2(a=10,b=20,c=30):
    print(a,b,c)

def fun3(a,b,c=None):
    print(a,b,c)


fun1(10,20,30)
fun2()
fun2(100,200)
fun2(b=400,c=500)
fun2(c=600)
fun2(a=400,c=900)
fun3(1,2)
fun3(10,20,c=400)
```

Output

```
10 20 30
10 20 30
100 200 30
10 400 500
10 20 600
400 20 900
1 2 None
10 20 400
```



Example:

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


si1=simple_int(9000,12)
si2=simple_int(10000,24,2.0)
print(f'si1={si1:.2f}')
print(f'si2={si2:.2f}')
```

Output

```
si1=1620.00
si2=4800.00
```

Example:

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

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



L1=[5,2,6,1,3,4]
print(f'Before Sorting {L1}')
sort(L1)
print(f'After Sorting {L1}')
sort(L1,reverse=True)
print(f'After Sorting {L1}')
```

Output

```
Before Sorting [5, 2, 6, 1, 3, 4]
After Sorting [1, 2, 3, 4, 5, 6]
After Sorting [6, 5, 4, 3, 2, 1]
```

Pass by reference

In python variables does not hold value, it hold address or reference of object. When function is called by sending object, it does not object but send object address or id.

In pass by reference any changes done using formal parameter it reflect actual argument/parameter.

Note: python does not support pass by value.