Function with variable length arguments or parameters

Function with variable length parameter receives 0 or more values or arguments.
Variable length parameters are 2 types

1. Function with variable length positional parameters
2. Function with variable length keyword parameters

Function with variable length positional parameters
This parameter receives values and store according their positions.
Variable length parameter is prefix with *
Variable length parameter is of type tuple
A function is defined with one variable length parameter
In application development function with variable length parameter is defined to perform aggregate operations. An operation required 0 or more values.

Syntax:

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

Example:

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



fun1(10)
fun1(10,20)
fun1()
fun1(10,1.5,1+2j,"naresh")
```

Output

```
(10,) <class 'tuple'>
(10, 20) <class 'tuple'>
() <class 'tuple'>
(10, 1.5, (1+2j), 'naresh') <class 'tuple'>
```

Example:

```
def maximum(*vargs):
    m=0
    for value in vargs:
        if value>m:
            m=value
    return m


res1=maximum()
print(res1)
res2=maximum(10,20)
print(res2)
res3=maximum(40,20,50,30,10)
print(res3)
```

Output

```
0
20
50
```

Example:

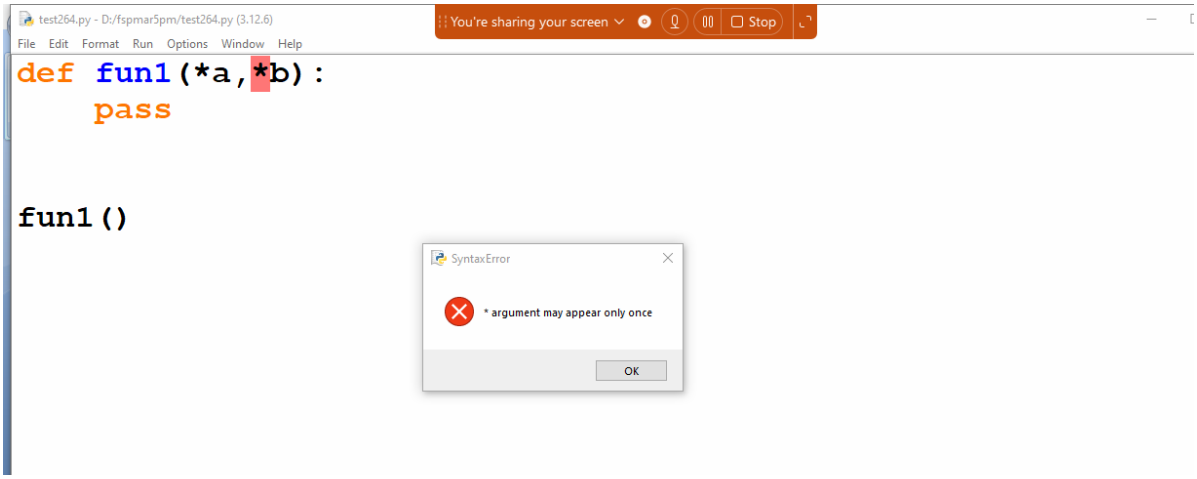
```
def sum_of_numbers(a,b,*c):
    t=a+b
    for value in c:
        t=t+value
    return t

res1=sum_of_numbers(10,20)
print(res1)
res2=sum_of_numbers(10,20,30,40,50)
print(res2)
```

Output

```
30
150
```

A function is defined with only one variable length parameter but it cannot have multiple variable length parameters.



Function with variable length keyword parameters

Function with variable length keyword parameter receives key and value.

1. Key → parameter name
2. Value → value of parameter

Function with variable length keyword parameter receives 0 or more values.

Function with variable length keyword parameter is prefix with **

Variable length keyword parameter is of type dictionary

Syntax:

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

Example:

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


def fun2(*a):
   print(a,type(a))


fun2(10,20,30,40)
fun1(x=10,y=20,z=30)
fun1()
```

```
fun2()
```

Output

```
(10, 20, 30, 40) <class 'tuple'>
{'x': 10, 'y': 20, 'z': 30} <class 'dict'>
{} <class 'dict'>
() <class 'tuple'>
```

Example:

```
def maximum(**kwargs):
    m=0
    for a in kwargs.items():
        if a[1]>m:
            t=a
            m=a[1]
    return t


res1=maximum(sub1=60,sub2=80,sub3=50)
res2=maximum(rohit=100,virat=50,surya=120)
print(res1)
print(res2)
```

Output

```
('sub2', 80)
('surya', 120)
```

Example:

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


fun1(10,20,30)
fun1(x=10,y=20,z=30)
fun1(10,20,x=10,y=20)
fun1()
```

Output

```
(10, 20, 30) {}
() {'x': 10, 'y': 20, 'z': 30}
(10, 20) {'x': 10, 'y': 20}
() {}
```

Example:

```
def fun1(a,d=None,*b,**c,):
    print(a,b,c,d)


fun1(100)
fun1(200,d=300)
fun1(100,200,300,400,500,600)
fun1(10,20,x=100,y=200)
fun1(1,2,3,4,x=5,y=6)
```

Output

```
100 () {} None
200 () {} 300
100 (300, 400, 500, 600) {} 200
10 () {'x': 100, 'y': 200} 20
1 (3, 4) {'x': 5, 'y': 6} 2
```

Nested Functions

A function defined inside function is called nested function or inner function.

Why nested functions?

1. Hiding function
2. Dividing functionality of function into sub functions
3. Developing special functions
    a. Decorators
    b. Generators

Syntax:

```
def  outer-function-name(parameter):
        statement-1
        statement-2
```

```
    def inner-function-name(parameter):
        statement-1
        statement-2
```

Points

1. Inner function can be used or invoked within outer function but not outside outer function
2. Inner function can access local variable of outer function, but outer function cannot access local variables of inner function

Example:

```
def fun1():
   print("Inside outer function")
   def fun2():
      print("Inside inner function")
   fun2()



fun1()
```

Output

```
Inside outer function
Inside inner function
```

Example:

```
def fun1():
   print("Inside outer function")
   def fun2():
      print("Inside inner function")
   return fun2



f2=fun1()
f2()
```

Output

```
Inside outer function
Inside inner function
```