Example:

```
# variable length arguments or parameters

def maximum(*values):
    if len(values)==0:
        return None
    elif len(values)==1:
        return values[0]
    else:
        m=0
        for value in values:
            if value>m:
                m=value
        return m



res1=maximum()
res2=maximum(10)
res3=maximum(10,5,6,20)
print(res1,res2,res3)
```

Output:

```
None 10 20
```

Example unpacking list values and sending to function, unpacking of list or iterable is done by using prefix *

```
# variable length arguments or parameters

def maximum(*values):
```

```
    if len(values)==0:
        return None
    elif len(values)==1:
        return values[0]
    else:
        m=0
        for value in values:
            if value>m:
                m=value
        return m



res1=maximum()
res2=maximum(10)
res3=maximum(10,5,6,20)
print(res1,res2,res3)
list1=[40,20,10,30,5]
res4=maximum(*list1) # unpacking list values and sending to
maximum function
print(res4)
```

Output:

```
None 10 20
40
```

Example of sum of n numbers:

```
def sum_of_numbers(*values,s=0):
    if len(values)==0:
        return s
    else:
        for value  in values:
```

```
        s=s+value
    return s


res1=sum_of_numbers()
res2=sum_of_numbers(10,20)
res3=sum_of_numbers(10,20,30,40,50,s=100)
print(res1,res2,res3,sep="\n")
```

Output

```
0
30
250
```

Example of defining order of parameters:

```
def fun1(a,b=10,*c): # required parameters,default parameters,variable length arguments
    print(a,b,c)

def fun2(a,*b,c=10): # required parameter, variable length parameter, default parameter
    print(a,b,c)

def fun3(*b,c=100,a): # variable length parameters, default parameters, required parameters
    print(b,c,a)


fun1(10)
fun1(100,200)
fun1(100,200,300,400,500,600)
fun2(10,20,30,40,50)
```

```
fun2(10,20,30,40,50,60,c=800)
fun2(100,c=900)
fun3(a=900)
fun3(100,200,300,a=600)
fun3(100,200,300,c=800,a=700)
fun3(100,200,300,400,a=900)
```

Output

```
10 10 ()
100 200 ()
100 200 (300, 400, 500, 600)
10 (20, 30, 40, 50) 10
10 (20, 30, 40, 50, 60) 800
100 () 900
() 100 900
(100, 200, 300) 100 600
(100, 200, 300) 800 700
(100, 200, 300, 400) 100 900
```

Function defined with parameter *, without name

If function having is having with parameters * without name, it work like separator. The parameters followed by * are called keyword parameters. When function is invoked values are given for these parameters using name.

Example:

```
def fun1(a,b,*,c,d): # 4 required parameters
   print(a,b,c,d)        #   a,b values are given using pos
                         #   c,d values are given using name(keyword)

fun1(10,20,c=30,d=40)
```

Output:
10 20 30 40

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


# fun1(10,20,30,40)
fun1(b=40)
```

Output:
Error

A function is defined with only one variable length argument or parameter (OR) Arbitrary Positional arguments or parameters.

Arbitrary parameters

1. Arbitrary Positional arguments or parameters
2. **Arbitrary Keyword arguments or parameters**

Arbitrary Keyword arguments or parameters (OR) keyword arguments

Function with arbitrary keyword parameters receives key and values.
Arbitrary parameters are prefix with ** followed by parameter name.
Arbitrary parameter is of type of dictionary.
A function is defined with only one keyword argument (OR) Arbitrary Keyword arguments.

Syntax:

def <function-name>(*vargs,kwargs):**

statement-1

statement-2

In application development function is defined with keyword arguments

1. When function required input in key and value pair
2. To manipulate content of dictionary

Example:

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



fun1()
fun1(a=10)
fun1(a=10,b=20,c=30)
fun1(x=10,y=20)
```

Output:

```
{} <class 'dict'>
{'a': 10} <class 'dict'>
{'a': 10, 'b': 20, 'c': 30} <class 'dict'>
{'x': 10, 'y': 20} <class 'dict'>
```