Example:

```
def sum_values(*values):
    s=0
    for value in values:
        s=s+value
    return s


res1=sum_values(10,20,30,40,50)
print(res1)
res2=sum_values()
print(res2)
res3=sum_values(10,1.5,2.5,20)
print(res3)
```

Output

```
150
0
34.0
```

Example:

```
def max_value(*values):
    if len(values)==0:
        return None
    elif len(values)>0:
        mvalue=values[0]
        for value in values:
            if value>mvalue:
                mvalue=value
        return mvalue

res1=max_value()
print(res1)
res2=max_value(10)
print(res2)
res3=max_value(10,50,30,70,20)
print(res3)
```

```
A=[10,50,60,20,30,5]
res4=max_value(*A)
print(res4)
str1="PYTHON"
res5=max_value(*str1)
print(res5)
```

Output

```
None
10
70
60
Y
```

Variable length keyword arguments

Variable length keyword argument receives more than one value; it receives values as key and value pair. Variable length keyword argument is prefix with **. This argument organizes data inside dictionary as key and value. A function defined with one variable length keyword argument.

Syntax:

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

Example:

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



fun1()
fun1(a=10,b=20)
fun1(rollno=1,name='naresh',course='python')
```

Output

```
{} <class 'dict'>
{'a': 10, 'b': 20} <class 'dict'>
{'rollno': 1, 'name': 'naresh', 'course': 'python'} <class 'dict'>
```

Example:

```
def max_values(*values,**kwargs):
    if len(values)>0:
        mvalue=values[0]
        for value in values:
            if value>mvalue:
                mvalue=value
        return mvalue
    elif len(kwargs)>0:
        A=list(kwargs.items())
        max_value=A[0]
        for t in A:
            if t[1]>max_value[1]:
                max_value=t
        return max_value
    else:
        return None

max_score=max_values(virat=50,rohit=100,surya=40)
print(max_score)
max_score=max_values(50,100,40)
print(max_score)
max_sales=max_values(jan=45000,feb=25000,mar=27000)
print(max_sales)
max_value=max_values()
print(max_value)
```

Output

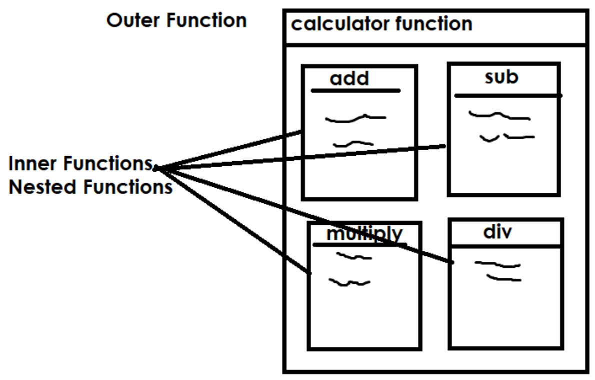
```
('rohit', 100)
100
('jan', 45000)
None
```

Example:
```
def print_dict(**kwargs):
    for k,v in kwargs.items():
        print(f'{k}-->{v}')



grade_dict={'naresh':'A',
            'suresh':'A',
            'ramesh':'B',
            'rajesh':'C',
            'kishore':'D'}
print_dict(**grade_dict)
```

Output
```
naresh-->A
suresh-->A
ramesh-->B
rajesh-->C
kishore-->D
```

Nested Functions
Defining function inside function is called nested function or inner function (OR) Writing function within function is called nested function.

Why nested functions?
1. Hiding Function
2. Special functions
    a. Decorator
    b. Closure
3. Dividing functionality of one function into sub functions.



Points to remember

1. Local Variables of outer function can access by inner function or nested function (OR) inner function can access local variables of outer function
2. Outer function cannot access local variables of inner function
3. Inner function cannot invoked outside outer function
4. Inner function is invoked within outer function.

Example:

```
def fun1(): # Outer Function
   print("fun1")
   def fun2(): # Inner Function
      print("fun2")
   fun2()

fun1()
```

Output

```
fun1
fun2
```

Python is an object oriented programming language and everything in python is represented as objects. A function is also object.

Example:

```
def fun1(): # Outer Function
   print("fun1")
```

```
    def fun2(): # Inner Function
        print("fun2")
    return fun2


a=fun1()
a()
```

Output

```
fun1
fun2
```

Example:

```
def fun1(): # Outer Function
    x=100 #Local Variable
    def fun2(): # Inner Function
        print(x)
    fun2()

def fun3(): # Outer Function
    def fun4(): # Inner Function
        x=200 # Local Variable
        return x
    x=fun4()
    print(x)

fun1()
fun3()
```

Output

```
100
200
```

Inner function can access local variable of outer function directly but it cannot assign or update value directly

```
def fun1(): # Outer Function
```

```
    x=100 # Local Variable
    def fun2(): # Inner Function
        x=200 #Local Variable of fun2
        print(x)
    fun2()
    print(x)


fun1()
```

Output

```
200
100
```

nonlocal

“nonlocal” is a keyword
Without using nonlocal keyword inner function can access local variable of outer function but it cannot update or modify. In order to update or modify inner function uses nonlocal keyword.

Syntax: nonlocal variable-list

After this statement, given list of variables are non local variables (local variables of outer function)