 # Example of a function which manipulate content of dictionary

```
def display(**kwargs):
    for key,value in kwargs.items():
        print(f'{key}--->{value}')



stud_dict={'naresh':'python',
           'suresh':'java',
           'ramesh':'oracle',
           'kishore':'c++'}

display(**stud_dict) # unpacking dictionary and send key and values
to display
```

Output:

```
naresh--->python
suresh--->java
ramesh--->oracle
kishore--->c++
```

Example:

```
def maximum(**kwargs):
    m=0
    k=None
    for key,value in kwargs.items():
        if value>m:
            m=value
            k=key
    return (k,m)
```

```
t1=maximum(a=10,b=20,c=30)
print(t1)
persons_dict={'naresh':50,'suresh':40,'ramesh':45}
t2=maximum(**persons_dict) # unpacking dictionry items
print(t2)
```

Output

```
('c', 30)
('naresh', 50)
```

Example:

```
def fun1(*vargs,**kwargs):
    for value in vargs:
        print(value)
    for key,value in kwargs.items():
        print(key,value)



fun1()
fun1(10,20,30,40,50)
fun1(a=10,b=20,c=30,d=40)
fun1(10,20,x=100,y=200)
```

Output:

```
10
20
30
40
```

```
50
a 10
b 20
c 30
d 40
10
20
x 100
y 200
```

Example:

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

def fun2(a,/,b=None):
    print(a,b)

fun1(10,b=20)
fun2(10)
fun2(10,b=20)
```

Output

```
10 20
10 None
10 20
```

Inner Function or Nested Function

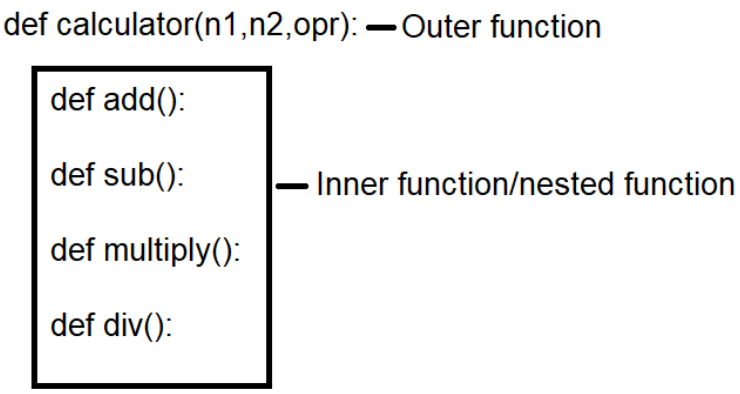
Function within function is called nested function or inner function (OR) defining function inside function is called nested function or inner function.

Use of nested functions

1. Developing special functions
    a. **Decorators**
    b. **Closures**
2. Hiding functionality of one function inside another function
3. Dividing functionality of one function into number of sub functions

Syntax:

def <function-name>([parameters]): Outer Function
    variables
    statements
    def <function-name>([parameters]): Inner Function
        variables
        statements



1. Inner function/nested function is invoked within outer function but cannot called/invoked outside outer function.

Example:

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

```
        print("inside inner function/nested function")
    fun2()



fun1()
```

Output

Inside outer function
inside inner function/nested function

2. Inner function can access local variables of outer function but outer function cannot access local variables of inner function

Example:

```
def fun1():
    x=100 # L.V.O.F
    print(f'Local variable of outer function {x}')
    def fun2():
        print(f'Local variable of outer function {x}') # Accessing local
variable of outer function

    def fun3():
        y=400 # Local variable of inner function

    fun2()
    print(y) # Error


fun1()
```

Output:

```
Local variable of outer function 100
Local variable of outer function 100
Traceback (most recent call last):
  File "E:/python7amdec23/funtest46.py", line 14, in <module>
    fun1()
  File "E:/python7amdec23/funtest46.py", line 11, in fun1
    print(y)
NameError: name 'y' is not defined
```

3. Inner function can access local variable of outer function directly but cannot modify or update value of local variable of outer function directly.

Example:

```
def fun1():
    x=100 # Local variable of outer function
    def fun2():
        x=400 # Create Local variable inside inner function
        print(f'Local variable of fun2 is {x}')
    fun2()
    print(f'Local variable of fun1 is {x}')

    def fun3():
        print(f'Local variable of fun1 is {x}')
    fun3()


fun1()
```

Output:

```
Local variable of fun2 is 400
```

Local variable of fun1 is 100
Local variable of fun1 is 100

nonlocal

nonlocal keyword is used to modify or update non local variable (a local variable of outer function) inside inner function.

Syntax:

nonlocal <variable-name>,<variable-name>,…

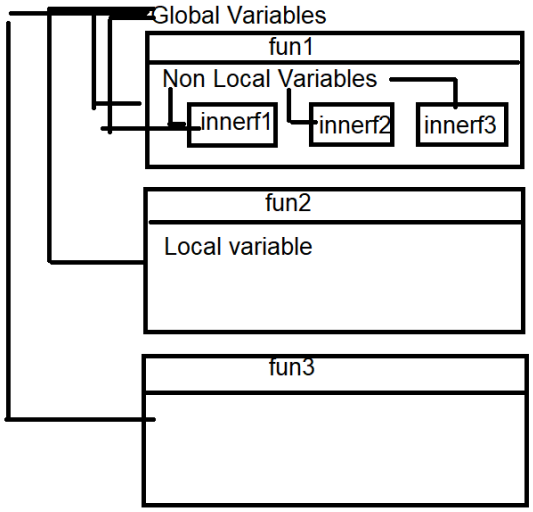
Example:

```
def fun1():
    x=100
    y=None
    print(f'Local variable of fun1 x={x}')
    def fun2():
        nonlocal x,y
        x=500
        y=600
        print(f'Local variable of fun1 x={x}')
        print(f'Local variable of fun1 y={y}')

    fun2()
    print(f'Local variable of fun1 x={x}')
    print(f'Local variable of fun1 y={y}')


fun1()
```

Output

Local variable of fun1 x=100

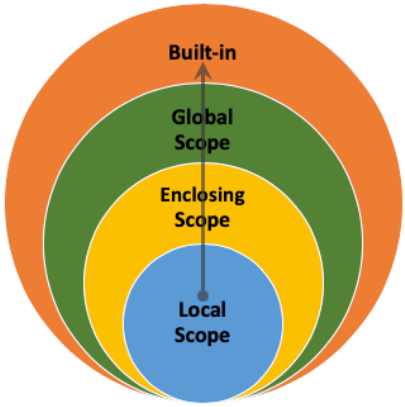
Local variable of fun1 x=500
Local variable of fun1 y=600
Local variable of fun1 x=500
Local variable of fun1 y=600



LEGB

The LEGB rule is a kind of name lookup procedure, which determines the order in which Python looks up names.

1. L Local
2. E Enclosed Block
3. G Global
4. B Built-ins Module



Example:

```
x=100 # Global Variable
def fun1():
    y=200 # Local variable
    def fun2():
        z=300 # Local variable
        print(x)
        print(y)
        print(z)
        print(__name__)
        print(pqr)
    fun2()


fun1()
```

Output:

```
100
200
300
__main__
Traceback (most recent call last):
```

```
  File "E:/python7amdec23/funtest49.py", line 14, in <module>
    fun1()
  File "E:/python7amdec23/funtest49.py", line 11, in fun1
    fun2()
  File "E:/python7amdec23/funtest49.py", line 10, in fun2
    print(pqr)
NameError: name 'pqr' is not defined
```