Nested Functions
A function within function is called nested function.

Need of nested functions

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

Syntax:
```
def <outer-function>([parameters]):
        statement-1
        def <inner-function>/<nested function>([parameters]):
                statement-2
                statement-3
        statement-4
        statement-5
```

Inner function is used inside outer function but cannot accessed outside outer function.

Example:
```python
def fun1():
    print("Outer function")
    def fun2():
        print("Inner function")
    fun2()


fun1()
```

Output:
```
Outer function
Inner function
```

Inner function can access the local variables of outer function or inner function performs operation using data of outer function.
Outer function cannot access local variables of inner function.

```
def fun1():
    x=100 # Local variable of fun1
    def fun2(): # nested function/inner function
        print(f'Local variable of fun1 x={x}')

    fun2()

fun1()
```

Output

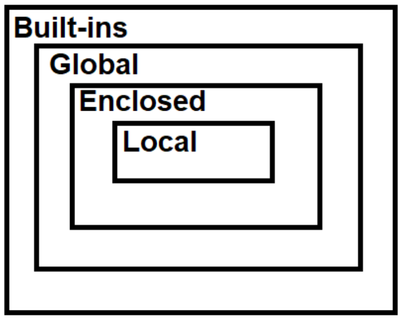
```
Local variable of fun1 x=100
```

What is LEGB rule in Python?

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

• For example, if we access a name, then Python will look that name up sequentially in the local, enclosing, global, and built-in scope. Local (or function) scope

L → Local
E → Enclosed
G → Global
B → Built-in



Example:

```
z=300 # G.V
def fun1():
    x=100 # Local variable of fun1
    def fun2(): # nested function/inner function
        y=200 # L.V of fun2
        print(f'Local variable of fun1 x={x}')
        print(f'Local variable of fun2 y={y}')
        print(f'Global variable z={z}')
    fun2()

fun1()
```

Output:

```
Local variable of fun1 x=100
Local variable of fun2 y=200
Global variable z=300
```

Nested function access local variable of outer function directly but cannot modify or update value directly.

Example:

```
def fun1():
    x=100 # Local variable of fun1
    def fun2(): # nested function/inner function
        x=200 # create local variable fun2
        print(x)
    fun2()
    print(x)

fun1()
```

Output:

```
200
100
```

nonlocal

This keyword is used to access local variable declared/created inside enclosed block. Using this keyword inner function can modify or update nonlocal variable (variable created inside enclosed scope).

Example

```
def fun1():
    x=100 # Local variable of fun1
    y=300
    def fun2(): # nested function/inner function
        nonlocal x
        x=200
        print(x)
        print(y)

    fun2()
    print(x)

fun1()
```

Output

```
200
300
200
```

Example:

```
def calculator(n1,n2,opr):
    res=None
    def add():
        nonlocal res
        res=n1+n2
    def sub():
        nonlocal res
        res=n1-n2
    def multiply():
        nonlocal res
        res=n1*n2
    def div():
        nonlocal res
        res=n1/n2
    if opr=='+':
        add()
    if opr=='-':
        sub()
    if opr=='*':
        multiply()
    if opr=='/':
        div()
    return res


def main():
    num1=int(input("Enter First Number "))
    num2=int(input("Enter Second Number"))
    opr=input("Enter Operator ")
    result=calculator(num1,num2,opr)
    print(f'Result is {result}')


main()
```

Output:

```
Enter First Number 5
Enter Second Number8
Enter Operator //
Result is None
```

Decorators