Inner function or nested function can perform operation using data of outer function.

Example:

```
def fun1():
    x=100 # local variable
    y=200 # local variable
    def fun2():
        print(x,y)
    fun2()


def main():
    fun1()

main()
```

Output:

```
100 200
```

Outer function cannot access local variables inner function

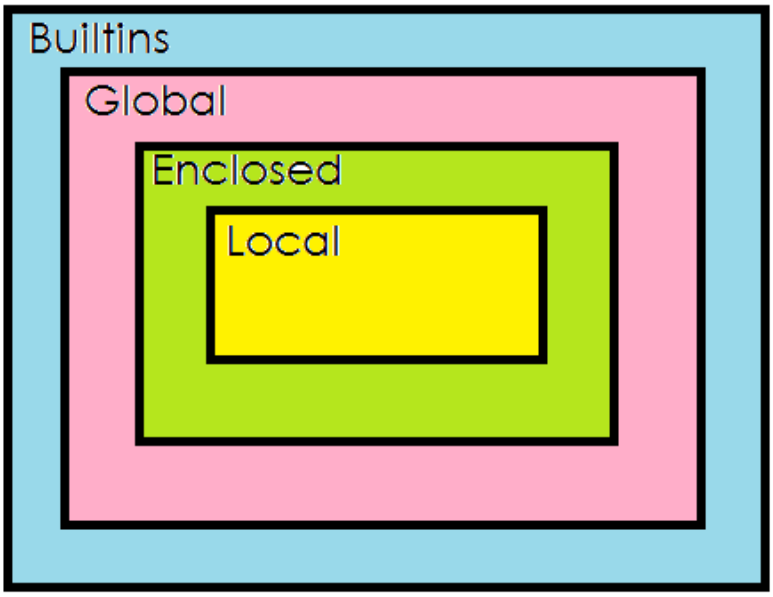
```
def fun1():
    def fun2():
        x=100 # local variable
        y=200 # local variable
    fun2()
    print(x,y)


fun1()
```

Output:

```
Traceback (most recent call last):
  File "F:/python6pmaug/funtest36.py", line 9, in <module>
    fun1()
  File "F:/python6pmaug/funtest36.py", line 6, in fun1
    print(x,y)
NameError: name 'x' is not defined
```

LEGB
LEGB stands,
L → Local
E → Enclosed Block
G → Global
B → Built-ins

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.




```
x=100
def fun1():
    y=200
    def fun2():
        z=300
        print(x)
        print(y)
        print(z)
        print(__doc__)

    fun2()
```

```
def main():
    fun1()

main()
```

Output:
100
200
300
None

Nested function or inner function can access local data of outer function but cannot modify or update data directly.

Example:
```
def fun1():
    x=100
    def fun2():
        x=200 # create LV
        print(x)
    fun2()
    print(x)

def main():
    fun1()

main()
```

Output:
200
100

nonlocal keyword
this keyword is used update or modify value of nonlocal variable (the variable declared inside enclosed block)

Example:
```
def fun1():
    x=100
    def fun2():
```

```
        nonlocal x
        x=200
        print(x)
    fun2()
    print(x)

def main():
    fun1()
main()
```

Output:

```
======== RESTART: F:/python6pmaug/funtest39.py =======
200
200
```

Example:

```
def calculator(n1,n2,opr):
    res=0
    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:

```
======== RESTART: F:/python6pmaug/funtest40.py =======
enter first number10
enter second number20
enter operator+
result is 30

======== RESTART: F:/python6pmaug/funtest40.py =======
enter first number10
enter second number5
enter operator*
result is 50
```

Decorator

Decorator is a special function in python.
A function returning another function, usually applied as a function transformation using the @wrapper syntax. Common examples for decorators are classmethod() and staticmethod().
Decorators are used to extend functionality of existing function without modifying it.
Decorator function receive input as one function and return another function as output.

Example:

```
def dfun2(f):
    def tfun1():
```

```
        f()
        print("new features")
    return tfun1

@dfun2
def fun1():
    print("inside fun1")

@dfun2
def fun3():
    print("inside fun3")

def main():
    fun1()
    fun3()
main()
```

Output:

```
inside fun1
new features
inside fun3
new features
```

Example:

```
def draw(f):
    def draw_line():
        print("*"*40)
        f()
        print("*"*40)
    return draw_line

@draw
def display():
    print("Welcome")


def main():
    display()
```

```
    '''d=draw(display) , d() '''
main()
```

Output:

```
****************************************
Welcome
****************************************
```

Example:

```
def newdiv(f):
    def smart_div(n1,n2):
        if n2==0:
            return 0
        else:
            return f(n1,n2)
    return smart_div
@newdiv
def div(n1,n2):
    return n1/n2


def main():
    num1=int(input("enter first number"))
    num2=int(input("enter second number"))
    res=div(num1,num2)
    print(f'result is {res:.2f}')

main()
```

Output:

```
======== RESTART: F:/python6pmaug/funtest43.py =======
enter first number5
enter second number2
result is 2.50

======== RESTART: F:/python6pmaug/funtest43.py =======
enter first number6
enter second number0
result is 0.00
```