Decorator Chaining
Using multiple decorators is called decorator chaining.
One decorator is wrapped with another decorator is called decorator chaining.

Example:

```
def dollar(f):
    def print_msg_stars_dollar():
        print("$"*20)
        f()
        print("$"*20)
    return print_msg_stars_dollar

def stars(f):
    def print_msg_stars():
        print("*"*20)
        f()
        print("*"*20)
    return print_msg_stars

@dollar
@stars
def print_msg():
    print("Python Language")

print_msg()
```

Output

```
$$$$$$$$$$$$$$$$$$$$
********************
Python Language
********************
$$$$$$$$$$$$$$$$$$$$
```

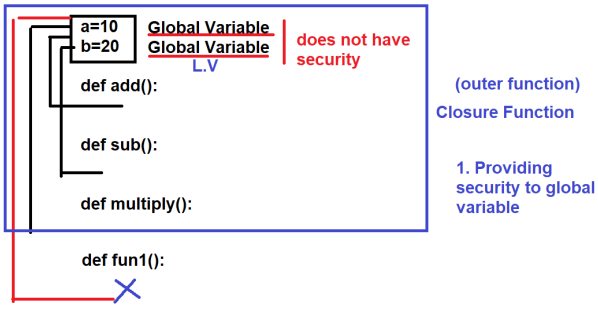
Example:

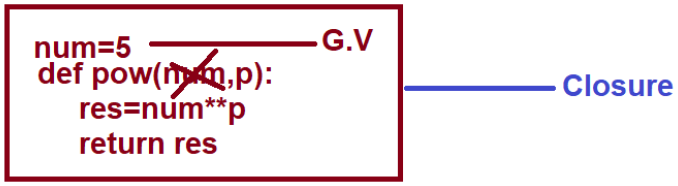
```
def upper(f):
    def print_str_upper(*iterable):
        for value in iterable:
            print(value.upper())
    return print_str_upper

@upper
def print_str(*iterable):
    for value in iterable:
        print(value)


print_str("naresh","suresh","kishore")
```

Output:
NARESH
SURESH
KISHORE

Closures

Closure is a special function in python.
Closure is developed using nested function.
Python closure is a nested function that allows us to access variables of the outer function even after the outer function is closed.
Closure in Python is an inner function object, a function that behaves like an object, that remembers and has access to variables in the local scope in which it was created even after the outer function has finished executing.





Basic steps for creating closure

1. Define a function with or without parameters (outer function)
2. Within function define inner function/closure, which access or bind data of outer function
3. Outer function returns closure function/inner function

Example:

```
def power(num):
    def find_pow(p): # closure function/inner function
        return num**p
    return find_pow


p5=power(5)
p6=power(6)
p4=power(4)
res1=p4(2)
res2=p4(3)
res3=p6(3)
res4=p6(5)
res5=p5(2)
res6=p5(3)
res7=p4(4)
print(res1,res2,res3,res4,res5,res6,res7)
```

Output:

16 64 216 7776 25 125 256

Example:

```
def filter_students(**kwargs):
    def filter_grade(grade):
        stud_grade = {key:value for key,value in kwargs.items() if
value==grade}
        return stud_grade
    return filter_grade


def main():
    student_data={'naresh':'A',
'suresh':'B','ramesh':'C','rajesh':'B','kiran':'A'}
    print(student_data)
    fg=filter_students(**student_data)
    studentA=fg('A')
    print(studentA)
    studentB=fg('B')
    print(studentB)
    studentC=fg('C')
    print(studentC)

main()
```

Output:

```
{'naresh': 'A', 'suresh': 'B', 'ramesh': 'C', 'rajesh': 'B', 'kiran': 'A'}
{'naresh': 'A', 'kiran': 'A'}
{'suresh': 'B', 'rajesh': 'B'}
{'ramesh': 'C'}
```

Example:

```
def calculator(n1,n2):
    def calculate(opr):
        if opr=='+':
            return n1+n2
        if opr=='-':
            return n1-n2
        if opr=='*':
            return n1*n2
```

```
        if opr=='/':
            return n1/n2
    return calculate


calc1=calculator(5,2)
r1=calc1('+')
r2=calc1('-')
calc2=calculator(6,3)
r4=calc2('+')
r5=calc2('-')
print(r1,r2,r4,r5)
```

Output:

```
7 3 9 3
```