Example:

```
def smart_division(f):
    def new_division(n1,n2):
        if n2==0:
            return 0
        else:
            return f(n1,n2)
    return new_division

@smart_division
def division(n1,n2):
    n3=n1/n2
    return n3

def main():
    a=int(input("enter  first number"))
    b=int(input("enter  second number"))
    c=division(a,b)
    print(a,b,c)

main()
```

Output:

```
enter  first number5
enter  second number0
5 0 0
>>>
```

Example

```
def fun3(function):
    def fun4():
        print("decorator")
        function()
    return fun4

@fun3
def fun1():
    print("fun1")
@fun3
def fun2():
    print("fun2")
```

```
def main():
    fun1()
    fun2()

main()
```

Output:

```
decorator
fun1
decorator
fun2
>>>
```

Closures

Closure is a special function

Closure function returns a function

Closure is an inner function which perform operation uses data of outer function

If you want to perform different operations using data of outer function then use closures

Syntax:

```
def outer_function([arg1,arg2,…]):
        def inner_function([arg1,arg2,..]):
                statement-1
                statement-2
        return inner function/closure-function
```

```
def find_power(num):
    def power(p):
      return num**p
    return power

p=find_power(2)
res1=p(1)
res2=p(2)
res3=p(3)

p1=find_power(3)
r1=p1(1)
r2=p1(2)
r3=p1(3)
r4=p(4)
```

$2^1 + 2^2 + 2^3$

```
def find_power(num):
    def power(p):
        return num**p
    return power
```

```
def main():
    pow1=find_power(2)
    res1=pow1(2)
    res2=pow1(3)
    print(res1,res2)
    pow2=find_power(3)
    r1=pow2(2)
    r2=pow2(3)
    print(r1,r2)

main()
```

Output:

```
4 8
9 27
>>>
```

Example:

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


def main():
    calc1=calculator(10,5)
    res1=calc1('+')
    res2=calc1('-')
    res3=calc1('*')
    print(res1,res2,res3)
    calc2=calculator(5,2)
    r1=calc2('*')
    r2=calc2('/')
```

```
    print(r1,r2)
main()
```

Output:

```
15 5 50
10 2.5
>>>
```

Example:

```
def draw_line(ch):
    def draw(l):
        for i in range(l):
            print(ch,end='')
        print()
    return draw

def main():
    draw1=draw_line('*')
    draw1(10)
    draw1(50)
    draw1(30)
    draw2=draw_line('$')
    draw2(5)
    draw2(10)
    draw1(50)
main()
```

Output:

```
**********
**************************************************
******************************
$$$$$
$$$$$$$$$$
**************************************************
>>>
```

Generators

What is generator?

A function which returns a generator iterator.

It looks like a normal function except that it contains yield expressions for producing a series of values usable in a for-loop or that can be retrieved one at a time with the next() function.

yield keyword
yield is a keyword
yield return value to caller pause execution of function
when iterator next() method is called it resumes back execute function