



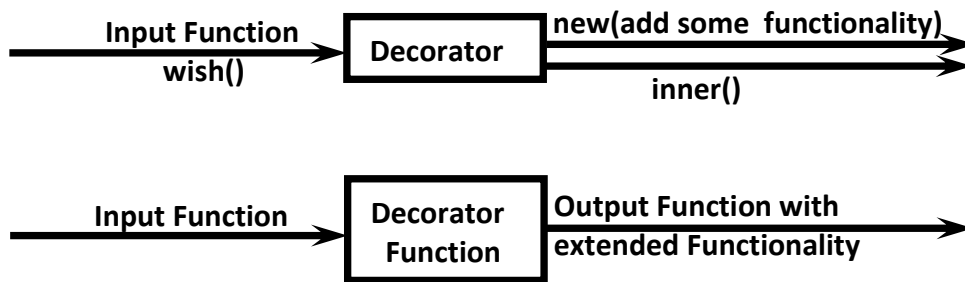
Learn Complete Python In Simple Way



DECORATOR FUNCTIONS STUDY MATERIAL



Decorator is a function which can take a function as argument and extend its functionality and returns modified function with extended functionality.



The main objective of decorator functions is we can extend the functionality of existing functions without modifies that function.

- 1) `def wish(name):`
- 2) `print("Hello",name,"Good Morning")`

This function can always print same output for any name

Hello Durga Good Morning
Hello Ravi Good Morning
Hello Sunny Good Morning

But we want to modify this function to provide different message if name is Sunny.
We can do this without touching wish() function by using decorator.

- 1) `def decor(func):`
- 2) `def inner(name):`
- 3) `if name=="Sunny":`
- 4) `print("Hello Sunny Bad Morning")`
- 5) `else:`
- 6) `func(name)`
- 7) `return inner`
- 8)
- 9) `@decor`
- 10) `def wish(name):`
- 11) `print("Hello",name,"Good Morning")`
- 12)
- 13) `wish("Durga")`
- 14) `wish("Ravi")`
- 15) `wish("Sunny")`



Output

Hello Durga Good Morning

Hello Ravi Good Morning

Hello Sunny Bad Morning

In the above program whenever we call wish() function automatically decor function will be executed.

How to call Same Function with Decorator and without Decorator:

We should not use @decor

```
1) def decor(func):
2)     def inner(name):
3)         if name=="Sunny":
4)             print("Hello Sunny Bad Morning")
5)         else:
6)             func(name)
7)     return inner
8)
9) def wish(name):
10)    print("Hello",name,"Good Morning")
11)
12) decorfunction=decor(wish)
13)
14) wish("Durga") #decorator wont be executed
15) wish("Sunny") #decorator wont be executed
16)
17) decorfunction("Durga")#decorator will be executed
18) decorfunction("Sunny")#decorator will be executed
```

Output

Hello Durga Good Morning

Hello Sunny Good Morning

Hello Durga Good Morning

Hello Sunny Bad Morning

```
1) def smart_division(func):
2)     def inner(a,b):
3)         print("We are dividing",a,"with",b)
4)         if b==0:
5)             print("OOPS...cannot divide")
6)             return
7)         else:
8)             return func(a,b)
```



```
9) return inner
10)
11) @smart_division
12) def division(a,b):
13)     return a/b
14) print(division(20,2))
15) print(division(20,0))
```

Without Decorator we will get Error. In this Case Output is:

10.0

Traceback (most recent call last):

File "test.py", line 16, in <module>

print(division(20,0))

File "test.py", line 13, in division

return a/b

ZeroDivisionError: division by zero

With Decorator we won't get any Error. In this Case Output is:

We are dividing 20 with 2

10.0

We are dividing 20 with 0

OOPS...cannot divide

None

```
1) def marriedecor(func):
2)     def inner():
3)         print('Hair decoration...')
4)         print('Face decoration with Platinum package')
5)         print('Fair and Lovely etc..')
6)         func()
7)     return inner
8)
9) def getready():
10)    print('Ready for the marriage')
11)
12) decorated_getready=marriedecor(getready)
13)
14) decorated_getready()
```

Decorator Chaining

We can define multiple decorators for the same function and all these decorators will form Decorator Chaining.



Eg:

@decor1

@decor

def num():

For num() function we are applying 2 decorator functions. First inner decorator will work and then outer decorator.

```
1) def decor1(func):
2)     def inner():
3)         x=func()
4)         return x*x
5)     return inner
6)
7) def decor(func):
8)     def inner():
9)         x=func()
10)        return 2*x
11)    return inner
12)
13) @decor1
14) @decor
15) def num():
16)     return 10
17)
18) print(num())
```