



PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Decorators

Prof. Sindhu R Pai

PCPS Theory Anchor - 2024

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



- A powerful and useful tool in Python since it allows programmers to **modify the behavior of function or class**.
- Decorators **wrap a function and modify its behavior in one or the other way, without changing the source code** of the function being decorated.
- In Decorators, **functions are taken as the argument into another function and then called inside the wrapper function**.

Function Decorators are used

- When we need to change the behavior of a function without modifying the function itself.
Eg: logging, test performance, verify permissions and so on.
- When we need to run the same code on multiple functions. This avoids writing duplicate code.

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



1. Example:

```
def func_decorator(func):  
    def inner_func():  
        print("Hello, before the function is called")  
        func()  
        print("Hello, after the function is called")  
    return inner_func
```

```
def func_hello():  
    print("Inside Hello function")
```

```
hello = func_decorator(func_hello) Refers to inner_func()  
hello()
```

Output:

```
Hello, before the function is called  
Inside Hello function  
Hello, after the function is called
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



- *func_decorator* is the decorator function, accepts another function as an argument and "decorates it".
- *func_hello* is an ordinary function that we need to decorate.
- *inner_func* is the wrapper function, that is actually decorating the *func_hello* function. In this example, all it does is print a simple statement before and after *func_hello*.

The function decorator in the above example can also be implemented in other way. (See Example 2)

By using @ symbol

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



2. Example (Same as Exampe1 with different format of Decorator):

```
def func_decorator(func):  
    def inner_func():  
        print("Hello, before the function is called")  
        func()  
        print("Hello, after the function is called")  
    return inner_func
```

```
@func_decorator  
def func_hello():  
    print("Inside Hello function")
```

```
func_hello()
```

Output:

```
Hello, before the function is called  
Inside Hello function  
Hello, after the function is called
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



3. Example

import math

```
def calculate(f):          #decorator function
    def inner1(*args):    #*args is variable length argument
        print("Decorator")
        f(*args)          # this is being decorated by decorator
        print("*****")
    return inner1
```

@calculate

```
def factorial(num):      #factorial() getting decorated
    print(math.factorial(num))
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



3. Example (contd...)

@calculate

```
def squarerooot(num):    #squarerooot() getting decorated
    print(math.sqrt(num))
```

@calculate

```
def maximum(*num):    #maximum() getting decorated
    print(max(num[0],num[1],num[2]))
```

```
factorial(5)           #calls decorated factorial()
squarerooot(16)         #calls decorated sqrt1()
maximum(23,9,78)        #calls decorated maximum()
```

Output:

Decorator

120

Decorator

4.0

Decorator

78

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



4. Example:

```
import math
```

```
def compute(func):      #decorator function
```

```
    def inner(a,b):
```

```
        print("Computing hypotenuse")
```

```
        func(a,b)      # this is being decorated by decorator
```

```
        print("*****")
```

```
    return inner
```

```
@compute
```

```
def hypotenuse(a, b):    # hypotenuse() is getting decorated
```

```
    h=math.sqrt(a*a+b*b)
```

```
    print(h)
```

```
hypotenuse(3,4)          #calls decorated hypotenuse
```

Output:

```
Computing hypotenuse
```

```
5.0
```

```
*****
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



Chaining Decorators - Decorating a function with multiple decorators.

```
def decorator_x(func):  
    def inner_func():  
        print("X"*20)    #Printing X 20 times  
        func()  
        print("X"*20)    #Printing X 20 times  
    return inner_func
```

```
def decorator_y(func):  
    def inner_func():  
        print("Y"*20)    #Printing Y 20 times  
        func()  
        print("Y"*20)    #Printing Y 20 times  
    return inner_func
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



```
def func_hello():  
    print("Hello")
```

```
hello = decorator_y(decorator_x(func_hello))  
hello()
```

#Chaining Decorators

Output:

```
YYYYYYYYYYYYYYYYYYYYYYY  
XXXXXXXXXXXXXXXXXXXXXXX  
Hello  
XXXXXXXXXXXXXXXXXXXXXXX  
YYYYYYYYYYYYYYYYYYYYYYY
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



Above Example can be implemented with different format of Decorators.

```
def decorator_x(func):
    def inner_func():
        print("X"*20)    #Printing X 20 times
        func()
        print("X"*20)    #Printing X 20 times
    return inner_func
```

```
def decorator_y(func):
    def inner_func():
        print("Y"*20)    #Printing Y 20 times
        func()
        print("Y"*20)    #Printing Y 20 times
    return inner_func
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



```
@decorator_y #Chaining Decorators
```

```
@decorator_x
```

```
def func_hello():  
    print("Hello")
```

```
func_hello()
```

Output:

```
YYYYYYYYYYYYYYYYYYYYYYY  
XXXXXXXXXXXXXXXXXXXXXXX  
Hello  
XXXXXXXXXXXXXXXXXXXXXXX  
YYYYYYYYYYYYYYYYYYYYYYY
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions - Decorators



Callback vs. Closure vs. Decorators

- **Callback** – a function that is passed as an argument to other function.
- **Closure** - a function object that remembers values in enclosing scopes even if they are not present in memory. It implements Data Encapsulation(Data hiding).
- **Decorators** - a way to modify the behavior of a function without directly changing its source code. It allows adding functionality to an existing function by wrapping it with another function.

Functions Decorators: Summary

- A Decorator is just a function that takes another function as an argument and extends its behavior without explicitly modifying it.
- Decorators allow us to wrap another function in order to extend the behavior of wrapped function, without permanently modifying it.
- Using decorators, we can extend the features of different functions in a common way.



THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CDSAML & CCBD, PESU

Prof. Sindhu R Pai – sindhurpai@pes.edu

Prof. Sowmya Shree P