@Techprolog

About

Decorators in Python

www.techprolog.com

Decorators
Implementation

There are 2 functions:

1. **my_decorator**
2. **disp**

Step: 1 Apply decorator (@my_decorator) before definition of disp() function

```
# Decorators - It is a 'function' that accepts
                    # a 'function as parameter'
                    # and 'returns' a function
```

```
def my_decorator(fun):
    def inner_fun():
        value = fun()
        return value+2
    return inner_fun

@my_decorator
def disp():
    return 10

print(disp())
```

Decorators

Step: 2 Since we have written decorator **(@my_decorator)** which is written before disp() function, python compiler will pass function **disp (disp())** as parameter i.e.,

my_decorator(fun()) --> my_decorator(disp()).

```
# Decorators - It is a 'function' that accepts
                    # a 'function as parameter'
                    # and 'returns' a function

def my_decorator(fun):
    def inner_fun():
        value = fun()
        return value+2
    return inner_fun

@my_decorator
def disp():
    return 10

print(disp())
```

disp() function Passed as Parameter to my_decorator()

REASON: Decorator is implemented before definition of disp()

Step: 2 call function

Step: 3 Python compiler **WILL NOT execute** **def inner_fun()** as we have not called the function i.e., only function is defined here.

```
# Decorators - It is a 'function' that accepts
                    # a 'function as parameter'
                    # and 'returns' a function

def my_decorator(fun):
    def inner_fun():
        value = fun()
        return value+2
    return inner_fun

@my_decorator
def disp():
    return 10

print(disp())
```

Step:3 Python compiler will skip executing inner_fun() as it is not called

Decorators

Step: 4 Next, python compiler will execute return inner_fun
Reason: Here inner_fun() is called

```
# Decorators - It is a 'function' that accepts
                    # a 'function as parameter'
                    # and 'returns' a function
```

```
def my_decorator(fun):
    def inner_fun():
        value = fun()
        return value+2
    return inner_fun

@my_decorator
def disp():
    return 10

print(disp())
```

Step4:
inner_fun() is called

Step: 5 Next, python compiler will execute return inner_fun
Reason: Here inner_fun() is called

```
# Decorators - It is a 'function' that accepts
                    # a 'function as parameter'
                    # and 'returns' a function
```

```
def my_decorator(fun):
    def inner_fun():
        value = fun()
        return value+2
    return inner_fun

@my_decorator
def disp():
    return 10

print(disp())
```

Step5:
Now, Python
Compiler executes
inner_fun()

Decorators

Step: 6 Python compiler will execute the paremeter fun()
NOTE: fun() -> will execute the value/function name it holds i.e., it executes disp()

fun() -> disp() will return value 10

```
# Decorators - It is a 'function' that accepts
                    # a 'function as parameter'
                    # and 'returns' a function

def my_decorator(fun):
    def inner_fun():
        value = fun()
        return value+2
    return inner_fun

@my_decorator
def disp():
    return 10

print(disp())
```

Step6:
Now, fun() will hold disp() which return value 10

Step: 7 inner_fun() will return the output value of parameter my_decorator

fun() i.e., disp() = 10
and returns 10+2 = 12

```
# Decorators - It is a 'function' that accepts
                    # a 'function as parameter'
                    # and 'returns' a function

def my_decorator(fun):
    def inner_fun():
        value = fun()
        return value+2
    return inner_fun

@my_decorator
def disp():
    return 10

print(disp())
```

Step7:
inner_fun() will return value 10+2 = 12

```
12
```