

Python decoradores

/via: <https://medium.com/@LuisMBaezCo/decoradores-con-clases-y-funciones-en-python-2fafb22dba43>

- <https://codigofacilito.com/articulos/decoradores-python>

Decorador mediante una función

```
def decorator(func):
    print("Decorator")
    return func

@decorator
def Hello():
    print("Hello World")

Hello()
# [Output]:
# Decorator
# Hello World
```

Decorador mediante dos funciones

```
#print_args: Función decoradora
def printArgs(func):
    def innerFunc(*args, **kwargs):
        print(args)
        print(kwargs)
        return func(*args, **kwargs)

    return innerFunc

#foobar: Función decorada
@printArgs
def foobar(x, y, z):
    return x * y + z

print(foobar(3, 5, z=10))
# (3, 5)
# {'z': 10}
# 25 = 3 * 5 + 10
```

Decorador simple mediante Clases

(se puede crear una función estática `@staticmethod` en una clase e invocarla de esa manera)

```
class Decorator(object):
    """Clase de decorador simple."""
    def __init__(self, func):
```

```
self.func = func

def __call__(self, *args, **kwargs):
    print('Antes de ser llamada la función.')
    retorno = self.func(*args, **kwargs)
    print('Despues de ser llamada la función.')
    print(retorno)
    return retorno

@Decorator
def function():
    print('Dentro de la función.')
    return "Retorno"

function()
# Antes de ser llamada la función.
# Dentro de la función.
# Despues de ser llamada la función.
# 'Retorno'
```

```
import types
print(isinstance(function, types.FunctionType))
# False
print(type(function))
# <class '__main__.Decorator'>
```

Decorando métodos de una clase

```
from types import MethodType

class Decorator(object):

    def __init__(self, func):
        self.func = func

    def __call__(self, *args, **kwargs):
        print('Dentro del Decorador.')
        return self.func(*args, **kwargs)

    def __get__(self, instance, cls):
        # Retorna un método si se llama en una instancia
        return self if instance is None else MethodType(self, instance)

class Test(object):
    @Decorator
    def __init__(self):
        print("Dentro de la función decorada")

a = Test()
# Dentro del Decorador.
# Dentro de la función decorada
```

Decorador con parámetros mediante clases

```
class MyDec(object):
    def __init__(self, flag):
        self.flag = flag

    def __call__(self, original_func):
        decorator_self = self
        def wrappee(*args, **kwargs):
            print('en decorador antes de wrapee ', decorator_self.flag)
            original_func(*args,**kwargs)
            print('en decorador despues de wrapee', decorator_self.flag)
        return wrappee

@MyDec(flag='foo de fa fa')
def bar(a,b,c):
    print('En bar(...) : ',a,b,c)

if __name__ == "__main__":
    bar(1, "Hola", True)
```

#Out:

```
# en decorador antes de wrapee  foo de fa fa
# en bar 1 Hola True
# en decorador despues de wrapee foo de fa fa
```

From:

<https://miguelangel.torresegea.es/wiki/> - miguel angel torres egea

Permanent link:

<https://miguelangel.torresegea.es/wiki/development:python:decorators?rev=1726138480>

Last update: **12/09/2024 03:54**

