Decoradores

Los decoradores son funciones que reciben una función y devuelve otra función. Los decoradores sirven para extender la funcionalidad de una función o también para agregar a una función una funcionalidad yuxtapuesta a la misma. Por ejemplo, puedo decorar una función con otra función que mide el tiempo que tarda en ejecutarse la función decorada e imprimiendo dicho tiempo en pantalla. No importa que es lo que haga la función decorada, siempre puedo medir el tiempo que tarda en ejecutarse.

```
def time_meter(f):
   def wrap(*args, **kwargs):
      ti = time.time()
      result = f(*args, **kwargs)
      tf = time.time()
      etime = tf-ti
      print("La función", f.__name__, "tardó:", etime, "segundos")
      return result
   return wrap

@time_meter
def suma(a,b):
   return a+b

>>> suma(1, 3)
La función suma tardó: 3.5762786865234375e-06 segundos
4
```

Los decoradores permiten hacer algunas acciones previos y/o posteriores a la ejecución de la función que se está decorando. Con lo cual, con el decorador se pueden manipular los argumentos antes de llamar a la función decorada y manipular el resultado generado por la función decorada.

Un decorador también puede recibir parámetros. De esta manera puede cambiar su comportamiento según el valor que venga en cada uno. Veamos un ejemplo:

```
def logger(debug=False):
   def _logger(func):
      def inner(*args, **kwargs):
         if debug:
            print ("Modo debug")
         for i, arg in enumerate(args):
            print ("arg %d:%s" % (i, arg))
         return func(*args, **kwargs)
      return inner
   return _logger

# Se agrega mensaje de modo debug al ejecutar la función suma
@logger(True)
def suma(a,b):
   return a+b

>>> suma(2,5)
Modo debug
arg 0:2
arg 1:5
7

# No se agrega mensaje de modo debug al ejecutar la función suma
@logger(False)
def suma(a,b):
   return a+b

>>> suma(2,5)
arg 0:2
arg 1:5
7
```

Como se puede ver en el ejemplo, para agregar la opción de que el decorador pueda recibir parámetros se debe agregar una función más anidada, el decorador propiamente dicho recibe los parámetros propios y la primer función anidada recibe la función decorada. La tercer función anidada es la tiene la lógica del decorador.

Por último, los decoradores pueden anidarse, es decir, una función puede estar decorada por muchos decoradores. Vemos un ejemplo donde anidamos los dos decoradores que creamos en los otros ejemplos aplicados a la función suma:

```
@logger(True)
@time_meter
def suma(a,b):
   return a+b

>>> suma(2,7)
Modo debug
arg 0:2
arg 1:7
La función suma tardó: 0.00019478797912597656 segundos
9

@time_meter
@logger(True)
def suma(a,b):
   return a+b

>>> suma(2,8)
Modo debug
arg 0:2
arg 1:8
La función inner tardó: 8.082389831542969e-05 segundos
10
```