T12_Decoradores

January 29, 2021

1 Python de cero a experto

Autor: Luis Miguel de la Cruz Salas

1.1 Pythonico es más bonito: Pensando como pythonista (intermedio)

1.1.1 Decoradores

- Se denomina decorador a la persona dedicada a diseñar el interior de oficinas, viviendas o establecimientos comerciales con criterios estéticos y funcionales.
- Un decorador es un objeto de Python usado para modificar una función.
- Los decoradores son herramientas bonitas y útiles de Python.

Ejemplo 1. Pornele una envoltura (*wrap*) para regalo a una función que imprime un texto.

[1]:
```
from termcolor import colored

# Esta función contiene funciones anidadas, las cuales son las que
# decoran a la función que se recibe como parámetro.
def miDecorador(f):

    # La función que hace el decorado.
    def envoltura():
        linea = '-' * 20
        print(colored('.'+ linea + '.','blue'))
        f()
        print(colored('.'+ linea + '.','green'))
    return envoltura

# Una función cualquiera.
def funcionX():
    print('{:^20}'.format('Hola mundo'))

# Ejecutando la función de manera normal.
funcionX()
```

```
# Decorando la función.
#funcionX = miDecorador(funcionX) # Funcion decorada

# Ahora se ejecuta la función decorada.
#funcionX()

# Otra manera de decorar una función.
#@miDecorador
#def funcionY():
#    print('{:<20}'.format('Hola Pythonistas'))

# La ejecución después del decorado.
#funcionY()
```

```
Hola mundo
```

Ejemplo 2. Calcular el seno y coseno de un número y *colorear* el resultado.

[2]:
```
from termcolor import colored

def miColoreador(f):

    def coloreado(x):
        res = colored('| ', 'blue')
        res += colored(f.__name__, 'red', attrs=['bold'])
        res += '(' + str(x) + ') = ' + str(f(x))
        n = len(res)
        linea = '-' * n

        print(colored('.'+ linea + '.','blue'))
        print(res)
        print(colored('.'+ linea + '.','green'))
    return coloreado

from math import sin, cos

sin = miColoreador(sin)
cos = miColoreador(cos)

for f in [sin, cos]:
    f(3.141596)
```

```
.-------------------------------------------------------------.
| sin(3.141596) = -3.5897932384626433e-06
.-------------------------------------------------------------.
.-------------------------------------------------------------.
| cos(3.141596) = -0.9999999999944008
.-------------------------------------------------------------.
```

Ejemplo 3. Decorar funciones con un número variable de argumentos.

[3]:
```
from random import random, randint, choice

def otroDecorador(f):
    def envoltura(*args, **kwargs):
        cadena = colored('| ', 'blue')
        cadena += colored(f.__name__, attrs=['bold'])
        cadena += '(' + colored(str(args),'red',attrs=['bold']) + ') = '
        linea = '-' * 10
        print(colored('.'+ linea + '.','blue'))
        res = f(*args, **kwargs)
        print(cadena, res)
        print(colored('.'+ linea + '.','green'))

    return envoltura

random = otroDecorador(random)
randint = otroDecorador(randint)
choice = otroDecorador(choice)

random()
randint(3, 8)
choice([4, 5, 6])
```

```
.----------.
| random(()) =  0.6108416883774596
.----------.
.----------.
| randint((3, 8)) =  6
.----------.
.----------.
| choice(([4, 5, 6],)) =  5
.----------.
```

Ejemplo 4. Crear un decorador que calcule el tiempo de ejecución de una función.

[4]:
```
import time

def crono(f):
    """
    Regresa el tiempo que toma en ejecutarse la funcion.
    """
    def tiempo():
        t1 = time.perf_counter()
        f()
        t2 = time.perf_counter()
        return 'Elapsed time: ' + str((t2 - t1)) + "\n"
```

```
    return tiempo

@crono
def miFuncion():
    numeros = []
    for num in (range(0, 10000)):
        numeros.append(num)
    print('\nLa suma es: ' + str((sum(numeros))))

print(miFuncion())
```

```
La suma es: 49995000
Elapsed time: 0.0029883399984100834
```

Ejemplo 5. Detener la ejecución por un tiempo antes que una función sea ejecutada.

[5]:
```
from time import sleep

def sleepDecorador(function):

    def duerme(*args, **kwargs):
        sleep(2)
        return function(*args, **kwargs)
    return duerme


@sleepDecorador
def imprimeNumero(num):
    return num

print(imprimeNumero(222))

for num in range(1, 6):
    print(imprimeNumero(num), end = ' ')

print('\n --> happy finish!')
```

```
222
1 2 3 4 5
 --> happy finish!
```

Ejemplo 6. Crear un decorador que cheque que el argumento de una función que calcula el factorial, sea un entero positivo.

[6]:
```python
def checaArgumento(f):
    def checador(x):
        if type(x) == int and x > 0:
            return f(x)
        else:
            raise Exception("El argumento no es un entero positivo")
    return checador

@checaArgumento
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

for i in range(1,10):
    print(i, factorial(i))

print(factorial(-1))
```

```
1 1
2 2
3 6
4 24
5 120
6 720
7 5040
8 40320
9 362880
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-6-361e6a8a4bdd> in <module>
     17     print(i, factorial(i))
     18
---> 19 print(factorial(-1))

<ipython-input-6-361e6a8a4bdd> in checador(x)
      4             return f(x)
      5         else:
----> 6             raise Exception("El argumento no es un entero positivo")
      7     return checador
      8

Exception: El argumento no es un entero positivo
```

Ejemplo 7. Contar el número de llamadas de una función

[7]:
```
def contadorDeLlamadas(func):
    def cuenta(*args, **kwargs):
        cuenta.calls += 1
        return func(*args, **kwargs)
    cuenta.calls = 0
    return cuenta

@contadorDeLlamadas
def suma(x):
    return x + 1

@contadorDeLlamadas
def mulp1(x, y=1):
    return x*y + 1

print(suma.calls)

for i in range(4):
    suma(i)

mulp1(1, 2)
mulp1(5)
mulp1(y=2, x=25)

print(suma.calls)
print(mulp1.calls)
```

```
0
4
3
```

Ejemplo 8. Decorar una función con diferentes saludos.

[8]:
```
def buenasTardes(func):
    def saludo(x):
        print("Hola, buenas tardes, ", end='')
        func(x)
    return saludo

def buenosDias(func):
    def saludo(x):
        print("Hola, buenos días, ", end='')
        func(x)
    return saludo

@buenasTardes
```

```
def mensaje1(hora):
    print("son las " + hora)

mensaje1("3 pm")

@buenosDias
def mensaje2(hora):
    print("son las " + hora)

mensaje2("8 am")
```

```
Hola, buenas tardes, son las 3 pm
Hola, buenos días, son las 8 am
```

Ejemplo 9. (decorador con parámetros) El ejemplo anterior se puede realizar como sigue:

[9]:
```
def saludo(expr):
    def saludoDecorador(func):
        def saludoGenerico(x):
            print(expr, end='')
            func(x)
        return saludoGenerico
    return saludoDecorador

@saludo("Hola, buenas tardes, ")
def mensaje1(hora):
    print("son las " + hora)

mensaje1("3 pm")

@saludo("Hola, buenos días, ")
def mensaje2(hora):
    print("son las " + hora)

mensaje2("8 am")

@saludo("     ")
def mensaje3(hora):
    print(" <--- en griego " + hora)

mensaje3(" :D ")
```

```
Hola, buenas tardes, son las 3 pm
Hola, buenos días, son las 8 am
      <--- en griego  :D
```

[ ]:
```

```