




Decoradores



Ayudantía 5
María José Hidalgo, Joaquín Tagle



¿Qué son?

Funcion  decorador Función + (Comportamiento y/o datos adicionales)



Retorna una nueva función

Utilidades

- Evita modificar el código original.
- Evita crear una nueva función con un nombre distinto.

Sintaxis: opción 1

```
def decorator(func_1):  
    def function(args):  
        #modifico func_1  
    return function
```

```
#Opción 1  
def f1(args):  
    #código f1  
    return something  
  
f1 = decorator(f1)  
print(f1( ))
```

Sintaxis: opción 2

```
def decorator(func_1):  
    def function(args):  
        #modifico func_1  
    return function
```

```
#Opción 2  
@decorator  
def f1(args):  
    #código f1  
    return something  
  
print(f1( ))
```

Aquí tenemos una función que saluda

```
def saludar(frase):  
    return frase  
  
print(saludar("Hola amigos"))
```

```
> Hola amigos
```

Una simple función que saluda.

Pero queremos que a veces pueda saludar más buena onda.

El decorador...

```
def buena_onda(saludar): # Este decorador asume que la función entregada  
                        # recibe solo un argumento.
```

```
    def new_function(frase):          # Esta es la nueva función a retornar.  
        frase = frase.upper( ) + "! :) " # Hago los cambios necesarios.  
        res = saludar(frase)          # Ejecuto saludar.
```

```
    return res
```

```
return new_function # Se retorna sin llamarla, más adelante se ejecuta
```

El cambio

```
> saludar = buena_onda(saludar)
```

```
> print(saludar("Hola amigos"))
```

```
> HOLA AMIGOS! :)
```


Jerarquía de Decoradores

```
def decorator_gen(parameters):  
    def decorator(function):  
        def new_function(arguments):  
            # Hacer algo con function  
            # Se puede usar parameters  
            res = function(arguments)  
            # Se puede hacer algo con res  
  
            return res  
        return new_function  
    return decorator
```

Este decorador formatea texto para dejarlo en formato html:

```
def header(function): # Decorador que asume que function recibe dos args.
    def new_function(string_1, string_2):
        resultado = function(string_1, string_2)
        resultado = "<h1> + resultado + <\h1>"
        return resultado
    return new_function
```

¿Qué pasa si queremos (en el caso anterior)
hacer otros tipos de tag en html?

Ahora tenemos una función que crea decoradores

```
def tag(tag_type): # Recibe la palabra que se quiere usar para crear el
                    # decorador de tags (el parámetro).
    def decorator(function):
        def new_function(string_1, string_2):
            resultado = function(string_1, string_2)
            resultado = "<{0}>{1}<\{0}>".format(tag_type, resultado)
            return resultado
        return new_function
    return decorator # Se retorna el decorador
```

Decorando clases :D

También podemos modificar el comportamiento de las clases con decoradores!

Importante:

- `getattr(class_name, attribute_name)`
- `setattr(class_name, attribute_name, attribute_value)`

Decorando clases :D

```
def decorador(Class):  
    #Modificar la clase  
    return Class
```

```
def decorador2(parameter):  
    def _decorador2(class):  
        #Modificar la clase  
        return class  
    return _decorador2
```

Decorando clases :D

2 formas de usar los decoradores

```
#Esto se ve poco elegante :/  
decorador_de_clase = decorador(parameter)  
MiClase = decorador_de_clase(MiClase)
```

```
#Más intuitivo y elegante :D  
@decorador(parameter)  
class MiClase:  
    pass
```

Algunos ejemplos.

```
def decorador(parameter):#¿Qué debería ser “parameter”?
```

```
    def decorador_clase(Clase):
```

```
        old_init = getattr(Clase, “__init__”)
```

```
        def new_init(self, *args, **kwargs):
```

```
            old_init(self, *args, **kwargs)
```

```
            setattr(Clase, “id”, next(parameter))
```

```
        setattr(Clase, “__init__”, new_init)
```

```
        return Clase
```

```
    return decorador_clase
```

```
#¿Cuál es el error en este decorador?
```


Algunos ejemplos.

```
def decorador(parameter):#¿Qué debería ser “parameter”?
    def decorador_clase(Clase):
        def menor(self, a):
            return getattr(self, parameter) < getattr(a, parameter)
        def igual(self, a):
            return getattr(self, parameter) == getattr(a, parameter)
        setattr(Clase, “__eq__”, igual)
        setattr(Clase, “__lt__”, menor)
        return Clase
    return decorador_clase
```

¿DUDAS DE LA
MATERIA?

Ejercicio 1 (examen 2016-2)

Implemente un decorador and que haga que una función retorne True, si su output cumple con ciertas condiciones dadas por una cantidad arbitraria de funciones.

```
@decorador(lambda x: x < 0, lambda x: x > -100)
def aumentar(num):
    return num + 1
print(aumentar(-10))
print(aumentar(1))
```

Debe imprimir True y luego False

Ejercicio 2

Implemente un decorador que reciba el nombre de un método, y lo convierta en un método privado.

```
@decorador("saludar")
class Persona:
    def saludar(self):
        print("Hola como estas")
    def despedir(self):
        print("Chao, me voy")
```

El método saludar debe imprimir "Soy privado", mientras que el metodo despedirse imprime "Chao me voy"