En borrador permanente
el blog de Juanjo Conti – abstracto, lúdico y digital

Decoradores en Python (I) – Introducción

Publicado el 11 de julio, 2008 por Juanjo

Este artículo es el primero de un plan de 3 artículos. Empezamos con una introducción a los decoradores en Python.

Funciones

Cómo todo en Python, las funciones son objetos. La forma más común de crear un objeto de tipo <function> es mediante el keyword def:

```
def saludo():
    print "Hola"
```

Al realizar esta definición, el cuerpo de la función es compilado pero no ejecutado y el objeto de tipo <function> es asociado al nombre 'saludo'. Mediante este nombre podemos referirnos al objeto:

```
>>> saludo
<function saludo at 0xb7d82fb4>
```

y utilizando la notación de paréntesis podemos llamar a (ejecutar) la función.

```
>>> saludo()
```

```
Hola
```

Una función puede tener parámetros (los parámetros son nombres a los que podemos referirnos en el cuerpo de la función):

```
def saludo2(nombre):
    print "Hola %s" % nombre
```

```
def saludo3(nombre, apellido):
    print "Hola %s %s" % (nombre, apellido)
```

Cuando llamamos a la función con argumentos (los argumentos son valores que en principio se asocian uno a uno a los parámetros de la función):

```
>>> saludo2("Ceci")
Hola Ceci
```

```
>>> saludo3("Ceci", "Pucci")
Hola Ceci Pucci
```

Los últimos n parámetros pueden tener valores por defecto, entonces estas definiciones y sus consiguientes ejecuciones son válidas:

```
def saludo4(nombre, apellido="Conti"):
    print "Hola %s %s" % (nombre, apellido)
```

```
>>> saludo4("Juanjo")
Hola Juanjo Conti
```

```
>>> saludo4("Juanjo", "Garau")
Hola Juanjo Garau
```

```
def saludo5(nombre="Juanjo", apellido="Conti"):
    print "Hola %s %s" % (nombre, apellido)
```

```
>>> saludo5()
Hola Juanjo Conti
```

```
>>> saludo5("Mary")
Hola Mary Conti
```

Los últimos n argumentos pueden ser argumentos nombrados, es decir utilizando el nombre de los parámetros con los que el argumento se debe asociar. En las siguientes ejecuciones se pueden ver ejemplos de esto:

```
def saludo6(tratamiento, nombre, apellido):
    print "Hola %s %s %s" % (tratamiento, nombre, apellido)
```

```
>>> saludo6("Sr.", apellido="Conti", nombre="Juanjo")
Hola Sr. Juanjo Conti
```

```
>>> saludo6("Sr.", "Juanjo", apellido="Conti")
Hola Sr. Juanjo Conti
```

Los parámetros de una función pueden terminar con *<nombre> (una tupla con los últimos argumentos posicionales) y/o **<nombre> (un diccionario con los últimos argumentos nombrados).

```
def saludo7(tratamiento, *args):
    print "Hola %s %s" % (tratamiento, " ".join(args))
```

```
>>> saludo7("Sr.", "Juanjo", "Conti")
Hola Sr. Juanjo Conti
```

```
>>> saludo7("Sr.", "Juanjo", "Conti", "Garau")
Hola Sr. Juanjo Conti Garau
```

Notemos que esta forma de definir una función es bastante útil cuando no sabemos el número de argumentos que se recibirán.
La siguiente es la forma más genérica de definir una función:

```
def saludo8(*args, **kwargs):
    pass
```

Decoradores

Un decorador es una función 'd' que recibe como argumento otra función 'a' y retorna una nueva función 'b'. La nueva función 'b' es la función 'a' decorada con 'd'.

Supongamos que queremos avisarle a un sistema de seguridad cada vez que se ejecutan las funciones abrir_puerta y cerrar_puerta. Para hacer una simplificación, el aviso simplemente será imprimir por un mensaje en la pantalla. Podemos escribir el siguiente 'decorador':

```
def avisar(f):
    def inner(*args, **kwargs):
        f(*args, **kwargs)
        print "Se ha ejecutado %s" % f.__name__
    return inner
```

Las siguientes son las funciones a decorar:

```
def abrir_puerta():
    print "Abrir puerta"

def cerrar_puerta():
    print "Cerrar puerta"
```

```
>>> abrir_puerta()
```

```
Abrir puerta
>>> cerrar_puerta()
Cerrar puerta
```

Y ahora solo nos limitamos a seguir la defición que di al principio de un decorador:

```
abrir_puerta = avisar(abrir_puerta)
cerrar_puerta = avisar(cerrar_puerta)
```

Listo!, ambas funciones han sido decoradas:

```
>>> abrir_puerta()
Abrir puerta
Se ha ejecutado abrir_puerta
>>> cerrar_puerta()
Cerrar puerta
Se ha ejecutado cerrar_puerta
```

Azúca sintáctica

En Python 2.3, la anterior era la forma de decorar una función. A partir de Python 2.4 se a añadido azúcar sintáctica al lenguaje que nos permite hacer lo mismo de esta forma:

```
@avisar
def abrir_puerta():
    print "Abrir puerta"

@avisar
def cerrar_puerta():
    print "Cerrar puerta"
```

Esta es una forma mucho más visual de hacerlo.

Encadenando decoradores

La decoración de funciones puede encadenarse. Para ejemplificarlo vamos a suponer ahora que solo usuarios autenticados en el sistema pueden ejecutar las funciones abrir_puerta y cerrar puerta.

Nuevamente hacemos una simplificación. Existe la variable AUTHENTICATED que indica el estado del usuario actual. Si el usuario no está autenticado y se intente ejecutar alguna de las funciones, una excepción es lanzada.

```
def autenticado(f):
    def inner(*args, **kwargs):
        if AUTHENTICATED:
            f(*args, **kwargs)
       else:
           raise Exception
    return inner
```

Luego, la definición de abrir_puerta y cerrar_puerta debería ser:

```
@autenticado
@avisar
def abrir_puerta():
    print "Abrir puerta"

@autenticado
@avisar
def cerrar_puerta():
    print "Cerrar puerta"
```

Con AUTHENTICATED = True:

```
>>> cerrar_puerta()
Cerrar puerta
```

Se ha ejecutado cerrar_puerta

Pero si AUTHENTICATED = False:

```
>>> cerrar_puerta()
Traceback (most recent call last):
File "<stdin> ", line 1, in <module>
File "<stdin> ", line 6, in inner
Exception
```

update: 2° entrega.

Acerca de Juanjo

Mi nombre es Juanjo Conti, vivo en Santa Fe y soy Ingeniero en Sistemas de Información. Mi lenguaje de programación de cabecera es Python; lo uso para trabajar, estudiar y jugar. Como hobby escribí algunos libros.

Ver todas las entradas por Juanjo →

Esta entrada fue publicada en Aprendiendo Python y etiquetada decoradores, Python. Guarda el enlace permanente.

Gonzalo • hace 6 años

Muy bueno. Lo marqué para tenerlo como ayudamemoria de decoradores :)

Responder • Compartir ›

Facundo Batista • hace 6 años

Pedagógicamente, deberías explicar antes de llegar a decoradores que es un "closure", sino metiendo la variable AUTENTICATED (que, btw, debería ser AUTHENTICATED) le complica la vida a la persona que está siguiendo esto.

Después lo veo piola. Obvio, faltan cosas, pero asumo que por eso pusiste que esta es la primera parte, ;)

Saludos!

Responder • Compartir ›

jjconti Moderador • hace 6 años

Gracias Facu por la sugerencia y la corrección.

Responder • Compartir ›

flipflop • hace 5 años

Hola Juanjo, muchas gracias, está muy bueno, lo único malo es que ahora me dieron ganas de leer las partes que siguen y no las encuentro :(, jejejejje, estarán algún día, cierto?

Responder • Compartir ›

jjconti Moderador • hace 4 años

El plan original eran 3 artículos. Pero ya tengo material para 4. Creo que se va a cerrar con 5 :)

Responder • Compartir ›

theletteru • hace 3 años

Felicidades por el post!

Responder • Compartir ›

Marcelo Martinovic • hace 3 años

Imposible mas claro !!! gracias !!!

Responder • Compartir ›

Br0th3r • hace 3 años

¡Genial el artículo! :-)

• Responder • Compartir ›

Ricardo Manríquez • hace 2 años

Muchas gracias, con esta aportación tuya he comprendido lo de los decoradores.

• Responder • Compartir ›

Helder Silva • hace 6 meses

Muito bom! Sou brasileiro, e não consegui um artigo que me fizesse entender a utilidade dos decoradores python. Mas graças ao seu artigo, está tudo mais claro agora. Muchas gracias, direto do Brasil! =]

1 • Responder • Compartir ›

Karlos Tz • hace 4 meses

disculpen se que es un blog acerca de python y me gusta mucho como explica Juanjo, pero necesito si alguien conoce el lenguaje c# decirme si estoy en lo correcto.

en python utilizamos un x decorador con una funcion

@deco
def foo():
pass

en c# he visto algo similar (igual y me equivoque)

[algo_que_no_entiendo]
class Cualquiera
{
[algo_mas_que_no_entiendo]

public un_tipo_cualquiera_de_dato foo()
{...}
}

bueno, alguien me puede explicar si son equivalentes
como se llama a eso (disculpen la vaguedad de mis expresiones)
y/o si alguien conoce una página que explique eso a detalle

de antemano gracias

• Responder • Compartir ›

En borrador permanente

Funciona con WordPress.