Tipos especiales de funciones

Funciones recursivas

Una función recursiva es aquella que al ejecutarse hace llamadas a ella misma. Por lo tento tenemos que tener "un caso base" que hace terminar el bucle de llamadas. Veamos un ejemplo:

```
>>> def factorial(numero):
...     if(numero == 0 or numero == 1):
...         return 1
...     else:
...         return numero * factorial(numero-1)
...
>>> factorial(5)
120
```

Funciones lambda

Las funciones lambda nos sirven para crear pequeñas funciones anónimas, de una sola línea sobre la marcha.

```
>>> cuadrado = lambda x: x**2
>>> cuadrado(2)
```

Como podemos notar las funciones lambda no tienen nombre. Pero gracias a que lambda crea una referencia a un objeto función, la podemos llamar.

```
>>> lambda x: x**2
<function <lambda> at 0xb74469cc>
>>>
>>> (lambda x: x**2)(3)
9
```

Otro ejemplo:

```
>>> pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
>>> pairs.sort(key=lambda pair: pair[1])
>>> pairs
[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]
```

Decoradores

Los decoradores son funciones que reciben como parámetros otras funciones y retornan como resultado otras funciones con el objetivo de alterar el funcionamiento original de la función que se pasa como parámetro. Hay funciones que tienen en común muchas funcionalidades, por ejemplo las de manejo de errores de conexión de recursos I/O (que se deben programar siempre que usemos estos recursos) o las de validación de permisos en las respuestas de peticiones de servidores, en vez de repetir el código de rutinas podemos abstraer, bien sea el manejo de error o la respuesta de peticiones, en una función decorador.

```
>>> def tablas(funcion):
...     def envoltura(tabla=1):
...         print('Tabla del %i:' %tabla)
...         print('-' * 15)
...         for numero in range(0, 11):
...             funcion(numero, tabla)
...         print('-' * 15)
...     return envoltura
...
>>> @tablas
... def suma(numero, tabla=1):
...     print('%2i + %2i = %3i' %(tabla, numero, tabla+numero))
...
>>> @tablas
... def multiplicar(numero, tabla=1):
...     print('%2i X %2i = %3i' %(tabla, numero, tabla*numero))

# Muestra la tabla de sumar del 1
suma()
# Muestra la tabla de sumar del 4
suma(4)
# Muestra la tabla de multiplicar del 1
multiplicar()
# Muestra la tabla de multiplicar del 10
multiplicar(10)
```

Funciones generadoras

Un generador es un tipo concreto de iterador. Es una función que permite obtener sus resultados paso a paso.

```
>>> def par(inicio,fin):
...   for i in range(inicio,fin):
...     if i % 2==0:
...       yield i

>>> datos = par(1,5)
>>> next(datos)
2
>>> next(datos)
4

>>> for i in par(20,30):
...   print(i,end=" ")
20 22 24 26 28

>>> lista_pares = list(par(1,10))
>>> lista_pares
[2, 4, 6, 8]
```