Funciones lambda

Si empiezo diciendo que las funciones o expresiones lambda sirven para crear funciones anónimas, posiblemente me diréis ¿qué me estás contando?, así que vamos a tomarlo con calma, pues estamos ante unas de las funcionalidades más potentes de Python a la vez que más confusas para los principiantes.

Una función anónima, como su nombre indica es una función sin nombre. ¿Es posible ejecutar una función sin referenciar un nombre? Pues sí, en Python podemos ejecutar una función sin definirla con **def**. De hecho son similares pero con una diferencia fundamental:

> *El contenido de una función lambda debe ser una única expresión en lugar de un bloque de acciones.*

Y es que más allá del sentido de función que tenemos, con su nombre y sus acciones internas, una función en su sentido más trivial significa realizar algo sobre algo. Por tanto podríamos decir que, mientras las funciones anónimas **lambda** sirven para realizar funciones simples, las funciones definidas con **def** sirven para manejar tareas más extensas.

Si deconstruimos una función sencilla, podemos llegar a una función lambda. Por ejemplo tomad la siguiente función para doblar un valor:

```
def doblar(num):
    resultado = num*2
    return resultado

doblar(2)
```

```
4
```

Vamos a simplificar el código un poco:

```
def doblar(num):
    return num*2
```

Todavía más, podemos escribirlo todo en una sola línea:

```
def doblar(num): return num*2
```

Esta notación simple es la que una función lambda intenta replicar, fijaros, vamos a convertir la función en una función anónima:

```
lambda num: num*2
```

```
<function __main__.<lambda>>
```

Aquí tenemos una función anónima con una entrada que recibe **num**, y una salida que devuelve **num * 2**.

Lo único que necesitamos hacer para utilizarla es guardarla en una variable y utilizarla tal como haríamos con una función normal:

```
doblar = lambda num: num*2

doblar(2)
```

```
4
```

Gracias a la flexibilidad de Python podemos implementar infinitas funciones simples.

Por ejemplo comprobar si un número es impar:

```
impar = lambda num: num%2 != 0

impar(5)
```

```
True
```

Darle la vuelta a una cadena utilizando slicing:

```
revertir = lambda cadena: cadena[::-1]

revertir("Hola")
```

```
'aloH'
```

Incluso podemos enviar varios valores, por ejemplo para sumar dos números:

```
sumar = lambda x,y: x+y

sumar(5,2)
```

```
7
```

Como véis podemos realizar cualquier cosa que se nos ocurra, siempre que lo podamos definir en una sola expresión.

A continuación veremos como explotar al máximo la función lambda utilizándola en conjunto con otras funciones como **filter()** y **map()**.

Última edición: 6 de Octubre de 2018