Tipo de datos: iterador y generador

Iterador

Un objeto iterable es aquel que puede devolver un iterador. Normalmente las colecciones que hemos estudiados son iterables. Un iterador me permite recorrer los elementos del objeto iterable.

Definición de iterador. Constructor iter

```
>>> iter1 = iter([1,2,3])
>>> type(iter1)
<class 'list_iterator'>
>>> iter2 = iter("hola")
>>> type(iter2)
<class 'str_iterator'>
```

Función next(), reversed()

Para recorrer el iterador, utilizamos la función `next()`:

```
>>> next(iter1)
1
>>> next(iter1)
2
>>> next(iter1)
3
>>> next(iter1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

La función `reversed()` devuelve un iterador con los elementos invertidos, desde el último al primero.

```
>>> iter2 = reversed([1,2,3])
>>> next(iter2)
3
>>> next(iter2)
2
>>> next(iter2)
1
>>> next(iter2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

El módulo itertools

El módulo itertools contiene distintas funciones que nos devuelven iteradores.

Veamos algunos ejemplos:

`count()`: Devuelve un iterador infinito.

```
>>> from itertools import count
>>> counter = count(start=13)
>>> next(counter)
13
>>> next(counter)
14
```

`cycle()`: devuelve una secuencia infinita.

```
>>> from itertools import cycle
>>> colors = cycle(['red', 'white', 'blue'])
>>> next(colors)
'red'
>>> next(colors)
'white'
>>> next(colors)
'blue'
>>> next(colors)
'red'
```

`islice()` : Retorna un iterador finito.

```
>>> from itertools import islice
>>> limited = islice(colors, 0, 4)
>>> for x in limited:
...   print(x)
white
blue
red
white
```

Generadores

Un generador es un tipo concreto de iterador. Es una función que permite obtener sus resultados paso a paso. Por ejemplo, hacer una función que cada vez que la llamemos nos de el próximo número par. Tenemos dos maneras de crear generadores:

1. Realizar una función que devuelva los valores con la palabra reservada `yield`. Lo veremos con profundidad cuando estudiemos las funciones.
2. Utilizando la sintaxis de las "list comprehension". Por ejemplo:

```
>>> iter1 = (x for x in range(10) if x % 2==0)
>>> next(iter1)
0
>>> next(iter1)
2
>>> next(iter1)
4
```