Iteradores

Los iteradores son tipos de datos que pueden ser recorridos secuencialmente mediante el uso del ciclo **for**.

Para esto, los objetos iterables deben saber responder los mensajes *__iter__* y *__next__*. El mensaje *__iter__* devuelve un iterador, que debe ser un objeto que sepa responder el mensaje *__next__*. El mensaje *__next__* devuelve el próximo elemento de la secuencia.

Veamos un ejemplo:

```
lista = [1,2,3,4,5]
for elemento in lista:
    print(elemento)
```

Las listas son objetos iterables. Lo primero que hace el ciclo **for** es llamar a la función *iter()* del contenedor u objeto iterable. Esta función devuelve un objeto iterador que define el método *__next__* que accede elementos en el contenedor de a uno por vez. Cuando no hay más elementos en el contenedor, la función *__next__* levanta una excepción del tipo *StopIteration* que le avisa al ciclo **for** que debe terminar.

Ahora que hemos visto cómo funciona internamente el ciclo **for**, es fácil crear nuestros propios iteradores para luego recorrerlos con el ciclo **for**.

Con lo cual, para crear un iterador alcanza definir una clase que defina los métodos *__iter__* y *__next__*. En el caso del método *__iter__*, si la misma clase tiene definido el método *__next__*, alcanza que devuelva **self** (es decir, que se devuelva a sí misma).

El método *__next__* debería contener la lógica de cómo acceder al siguiente elemento de la secuencia.

Cabe destacar que los métodos *__iter__* y *__next__* son métodos mágicos. Los métodos mágicos son aquellos que comienzan y terminan con doble guión bajo, que no están pensados para ser invocados manualmente sino que son llamados por Python en situaciones particulares. En este caso el *__iter__* y el *__next__* son llamados por el **for**.

Veamos un ejemplo:

Si queremos hacer un iterador que recorra los elementos de una lista de atrás para adelante, podemos crearlo de la siguiente manera:

```
class Reversa:
    """Iterador para recorrer una secuencia de atrás para adelante."""

    def __init__(self, datos):
        self.datos = datos
        self.indice = len(datos)

    def __iter__(self):
        return self

    def __next__(self):
        if self.indice == 0:
            raise StopIteration()
        self.indice = self.indice - 1
        return self.datos[self.indice]
```

Con lo cual, si hago:

```
for elemento in Reversa([1,2,3,4]):
    print(elemento)
```

Se imprimirá en pantalla primero el elemento 4, seguido del elemento 3, luego el elemento 2 y por último el elemento 1.

De esta manera puedo crear mis propios objetos iterables con el comportamiento que necesite definir.