Iteradores

Iteradores

- Cuando no queremos iterar de forma secuencial con enteros sino con estructuras de datos tenemos que indicar al programa como hacerlo.
- Tenemos dos formas de iterar, mediante funciones generadoras o definiendo la forma de iterar en la definición de las clases.

Funciones generadoras

- La diferencia entre una función normal y otra generadora es el uso de ‘yield’ en vez de ‘return’. yield también devuelve valores pero, a diferencia de return, no detiene la función.

```
def cuadrados(n):
    x = 0
    while x < n:
        yield x**2
        x += 1
for i in cuadrados(10):
    print(i)
```

Generadores

- Las funciones generadoras cuando son llamadas devuelven objetos generadores.
- Estos objetos almacenan los valores de la iteración hasta que que se acaba.
- Podemos iterar fuera de los bucles for con la función next().

```
gen = cuadrados(5)

print(next(gen))
```

Generadores

- Cuando un iterador termina, al volver a llamar la función next(), lanza un error ‘StopIteration’.
- De hecho, un for no es más que la forma resumida de esto:

```
gen = iter(cuadrados(10)) # iter es redundante aquí.

while True:
    try:
        i = next(gen)
        print(i)
    except StopIteration:
        break
```

Definir iteradores

- Podemos definir en cualquier clase la forma de iterar. Para ello tenemos que definir los métodos __iter__ y __next__ (que son llamados por iter() y next() respectivamente).

```
class PowTwo:
  def __init__(self, max = 0):
    self.max = max
  def __iter__(self):
    self.n = 0
    return self   ← Debe devolverse a sí mismo o un generador.
  def __next__(self):
    if self.n <= self.max:
      out = 2 ** self.n
      self.n += 1
      return out
    else:
      raise StopIteration ← Debe de lanzar StopIteration en algún momento.
```