### 4.2.8. Rotura de bucles: break

El último programa diseñado aborta su ejecución tan pronto sabemos que el número estudiado no es primo. La variable *creo_que_es_primo* juega un doble papel: «recordar» si el número es primo o no al final del programa y abortar el bucle **while** tan pronto sabemos que el número no es primo. La condición del **while** se ha complicado un poco para tener en cuenta el valor de *creo_que_es_primo* y abortar el bucle inmediatamente.

Hay una sentencia que permite abortar la ejecución de un bucle desde cualquier punto del mismo: **break** (en inglés significa «romper»). Observa esta nueva versión del mismo programa:

es_primo_17.py — es_primo.py

```
1  num = int(raw_input('Dame un número: '))
2
3  creo_que_es_primo = True
4  divisor = 2
5  while divisor < num:
6      if num % divisor == 0:
7          creo_que_es_primo = False
8          break
9      divisor += 1
10
11 if creo_que_es_primo:
12     print 'El número', num, 'es primo'
13 else:
14     print 'El número', num, 'no es primo'
```

Cuando se ejecuta la línea 8, el programa sale inmediatamente del bucle, es decir, pasa a la línea 10 sin pasar por la línea 9.

Nuevamente estamos ante una comodidad ofrecida por el lenguaje: la sentencia **break** permite expresar de otra forma una idea que ya podía expresarse sin ella. Sólo debes considerar la utilización de **break** cuando te resulte cómoda. No abuses del **break**: a veces, una condición bien expresada en la primera línea del bucle **while** hace más legible un programa.

La sentencia **break** también es utilizable con el bucle **for-in**. Analicemos esta nueva versión de `es_primo.py`:

es_primo_18.py — es_primo.py

```
1  num = int(raw_input('Dame un número: '))
2
3  creo_que_es_primo = True
4  for divisor in range(2, num):
5      if num % divisor == 0:
6          creo_que_es_primo = False
7          break
8
9  if creo_que_es_primo:
10     print 'El número', num, 'es primo'
11 else:
12     print 'El número', num, 'no es primo'
```

### *Versiones eficientes de «se cumple para alguno/se cumple para todos»*

Volvemos a visitar los problemas de «se cumple para alguno» y «se cumple para todos». Esta vez vamos a hablar de cómo acelerar el cálculo gracias a la sentencia **break**.

Si quieres comprobar si una condición se cumple para todos los elementos de un conjunto y encuentras que uno de ellos no la satisface, ¿para qué seguir? ¡Ya sabemos que no se cumple para todos!

```
creo_que_se_cumple_para_todos = True
for elemento in conjunto:
    if not condición:
        creo_que_se_cumple_para_todos = False
        break

if creo_que_se_cumple_para_todos:
    print 'Se cumple para todos'
```

Como ves, esta mejora puede suponer una notable aceleración del cálculo: cuando el primer elemento del conjunto no cumple la condición, acabamos inmediatamente. Ese es el mejor de los casos. El peor de los casos es que todos cumplan la condición, pues nos vemos obligados a recorrer todos los elementos del conjunto. Y eso es lo que hacíamos hasta el momento: recorrer todos los elementos. O sea, en el peor de los casos, hacemos el mismo esfuerzo que veníamos haciendo para todos los casos. ¡No está nada mal!

Si quieres comprobar si una condición se cumple para alguno de los elementos de un conjunto y encuentras que uno de ellos la satisface, ¿para qué seguir? ¡Ya sabemos que la cumple alguno!

```
creo_que_se_cumple_para_alguno = False
for elemento in conjunto:
    if condición:
        creo_que_se_cumple_para_alguno = True
        break

if creo_que_se_cumple_para_alguno:
    print 'Se cumple para alguno'
```

Podemos hacer la misma reflexión en torno a la eficiencia de esta nueva versión que en el caso anterior.

Esta versión es más concisa que la anterior (ocupa menos líneas) y, en cierto sentido, más elegante: el bucle **for-in** expresa mejor la idea de que *divisor* recorre ascendentemente un rango de valores.

EJERCICIOS

▶ 131 Haz una traza del programa para el valor 125.

▶ 132 En realidad no hace falta explorar todo el rango de números entre 2 y $n-1$ para saber si un número $n$ es o no es primo. Basta con explorar el rango de números entre 2 y la parte entera de $n/2$. Piensa por qué. Modifica el programa para que sólo exploremos ese rango.

▶ 133 Ni siquiera hace falta explorar todo el rango de números entre 2 y $n/2$ para saber si un número $n$ es o no es primo. Basta con explorar el rango de números entre 2 y la parte entera de $\sqrt{n}$. (Créetelo.) Modifica el programa para que sólo exploremos ese rango.

▶ 134 Haz un programa que vaya leyendo números y mostrándolos por pantalla hasta que el usuario introduzca un número negativo. En ese momento, el programa mostrará un mensaje de despedida y finalizará su ejecución.

▶ 135 Haz un programa que vaya leyendo números hasta que el usuario introduzca un número negativo. En ese momento, el programa mostrará por pantalla el número mayor de cuantos ha visto.