5.2.2. Comparación de listas

Los operadores de comparación también trabajan con listas. Parece claro cómo se comportarán operadores como el de igualdad (==) o el de desigualdad (!=):

- si las listas son de talla diferente, resolviendo que las listas son diferentes;
- y si miden lo mismo, comparando elemento a elemento de izquierda a derecha y resolviendo que las dos listas son iguales si todos sus elementos son iguales, y diferentes si hay algún elemento distinto.

Hagamos un par de pruebas con el intérprete de Python:

```
>>> [1, 2, 3] == [1, 2] ↲
False
>>> [1, 2, 3] == [1, 2, 3] ↲
True
>>> [1, 2, 3] == [1, 2, 4] ↲
False
```

Los operadores <, >, <= y >= también funcionan con listas. ¿Cómo? Del mismo modo que con las cadenas, pues al fin y al cabo, tanto cadenas como listas son *secuencias*. Tomemos, por ejemplo, el operador < al comparar las listas [1, 2, 3] y [1, 3, 2], es decir, al evaluar la expresión [1, 2, 3] < [1, 3, 2]. Se empieza por comparar los primeros elementos de ambas listas. Como no es cierto que 1 < 1, pasamos a comparar los respectivos segundos elementos. Como 2 < 3, el resultado es *True*, sin necesidad de efectuar ninguna comparación adicional.

EJERCICIOS

▶ 218 ¿Sabrías decir que resultados se mostrarán al ejecutar estas sentencias?

```
>>> [1, 2] < [1, 2] ↲
>>> [1, 2, 3] < [1, 2] ↲
>>> [1, 1] < [1, 2] ↲
>>> [1, 3] < [1, 2] ↲
>>> [10, 20, 30] > [1, 2, 3] ↲
>>> [10, 20, 3] > [1, 2, 3] ↲
>>> [10, 2, 3] > [1, 2, 3] ↲
>>> [1, 20, 30] > [1, 2, 3] ↲
>>> [0, 2, 3] <= [1, 2, 3] ↲
>>> [1] < [2, 3] ↲
>>> [1] < [1, 2] ↲
>>> [1, 2] < [0] ↲
```

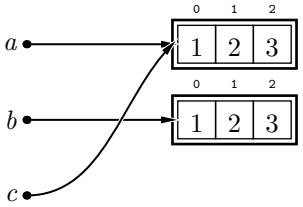
▶ 219 Diseña un programa que tras asignar dos listas a sendas variables nos diga si la primera es menor que la segunda. No puedes utilizar operadores de comparación entre listas para implementar el programa.

5.2.3. El operador is

Hemos visto que las listas conllevan una forma de reservar memoria curiosa: en ocasiones, dos variables apuntan a una misma zona de memoria y en ocasiones no, incluso cuando los datos de ambas variables son idénticos. Fíjate en este ejemplo:

```
>>> a = [1, 2, 3] ↲
>>> b = [1, 2, 3] ↲
>>> c = a ↲
```

Ya hemos visto que, tras efectuar las asignaciones, la memoria quedará así:



¿Qué ocurre si comparamos entre sí los diferentes elementos?

```
>>> a == b ↲
True
>>> a == c ↲
True
```

Efectivamente: siempre dice que se trata de listas iguales, y es cierto. Sí, pero, ¿no son «más iguales» las listas *a* y *c* que las listas *a* y *b*? A fin de cuentas, tanto *a* como *c* apuntan exactamente a la misma zona de memoria, mientras que *b* apunta a una zona distinta. Python dispone de un operador de comparación especial que aún no te hemos presentado: **is** (en español, «es»). El operador **is** devuelve *True* si dos objetos son en realidad el mismo objeto, es decir, si residen ambos en la misma zona de memoria, y *False* en caso contrario.

```
>>> a is b ↲
False
>>> a is c ↲
True
```

Python reserva nuevos bloques de memoria conforme evalúa expresiones. Observa este ejemplo:

```
>>> a = [1, 2] ↲
>>> a is [1, 2] ↲
False
>>> a == [1, 2] ↲
True
```

La segunda orden compara la lista almacenada en *a*, que se creó al evaluar una expresión en la orden anterior, con la lista `[1, 2]` que se crea en ese mismo instante, así que **is** nos dice que ocupan posiciones de memoria diferentes. El operador `==` sigue

devolviendo el valor *True*, pues aunque sean objetos diferentes son equivalentes elemento a elemento.

EJERCICIOS

▶ 220 ¿Qué ocurrirá al ejecutar estas órdenes Python?

```
>>> a = [1, 2, 3] ↲
>>> a is a ↲
>>> a + [] is a ↲
>>> a + [] == a ↲
```

▶ 221 Explica, con la ayuda de un gráfico que represente la memoria, los resultados de evaluar estas expresiones:

```
>>> a = [1, 2, 1] ↲
>>> b = [1, 2, 1] ↲
>>> (a[0] is b[0]) and (a[1] is b[1]) and (a[2] is b[2]) ↲
True
>>> a == b ↲
True
>>> a is b ↲
False
```

▶ 222 ¿Qué ocurrirá al ejecutar estas órdenes Python?

```
>>> [1, 2] == [1, 2] ↲
>>> [1, 2] is [1, 2] ↲
>>> a = [1, 2, 3] ↲
>>> b = [a[0], a[1], a[2]] ↲
>>> a == b ↲
>>> a is b ↲
>>> a[0] == b[1] ↲
>>> b is [b[0], b[1], b[2]] ↲
```

▶ 223 Que se muestra por pantalla como respuesta a cada una de estas sentencias Python:

```
>>> a = [1, 2, 3, 4, 5] ↲
>>> b = a[1:3] ↲
>>> c = a ↲
>>> d = a[:] ↲
>>> a == c ↲
>>> a == d ↲
>>> c == d ↲
>>> a == b ↲
>>> a is c ↲
>>> a is d ↲
>>> c is d ↲
>>> a is b ↲
```