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:

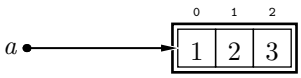
```
>>> a = [1, 2, 3, 4, 5] ↲
>>> b = a[1:3] ↲
>>> c = a ↲
>>> d = a[:] ↲
>>> a == c ↲
>>> a == d ↲
>>> c == d ↲
>>> a == b ↲
>>> a is c ↲
>>> a is d ↲
>>> c is d ↲
>>> a is b ↲
```

5.2.4. Modificación de elementos de listas

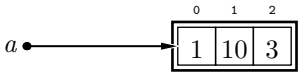
Hasta el momento hemos aprendido a crear listas y a consultar su contenido, bien accediendo a uno cualquiera de sus elementos (mediante indexación), bien recorriendo todos sus elementos (con un bucle **for–in**). En este apartado veremos cómo modificar el contenido de las listas.

Podemos asignar valores a elementos particulares de una lista gracias al operador de indexación:

```
>>> a = [1, 2, 3] ↲
```




```
>>> a[1] = 10 ↲
```




```
>>> a ↲
[1, 10, 3]
```

Cada celda de una lista es, en cierto modo, una variable autónoma: podemos almacenar en ella un valor y modificarlo a voluntad.

EJERCICIOS

▶ 224 Haz un programa que almacene en una variable *a* la lista obtenida con *range*(1,4) y, a continuación, la modifique para que cada componente sea igual al cuadrado del componente original. El programa mostrará la lista resultante por pantalla.

▶ 225 Haz un programa que almacene en *a* una lista obtenida con *range*(1,*n*), donde *n* es un entero que se pide al usuario y modifique dicha lista para que cada componente sea igual al cuadrado del componente original. El programa mostrará la lista resultante por pantalla.

▶ 226 Haz un programa que, dada una lista *a* cualquiera, sustituya cualquier elemento negativo por cero.

▶ 227 ¿Qué mostrará por pantalla el siguiente programa?

copias_2.py — copias.py

```
1  a = range(0, 5)
2  b = range(0, 5)
3  c = a
4  d = b[:]
5  e = a + b
6  f = b[:1]
7  g = b[0]
8  c[0] = 100
9  d[0] = 200
10 e[0] = 300
11 print a, b, c, d, e, f, g
```

Comprueba con el ordenador la validez de tu respuesta.

5.2.5. Mutabilidad, inmutabilidad y representación de la información en memoria

Python procura no consumir más memoria que la necesaria. Ciertos objetos son inmutables, es decir, no pueden modificar su valor. El número 2 es siempre el número 2. Es un objeto inmutable. Python procura almacenar en memoria una sola vez cada valor inmutable. Si dos o más variables contienen ese valor, sus referencias apuntan a la misma zona de memoria. Considera este ejemplo: