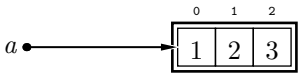
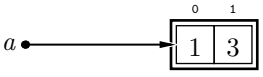
5.2.8. Borrado de elementos de una lista

También podemos eliminar elementos de una lista. Para ello utilizamos la sentencia **del** (abreviatura de «delete», que en inglés significa *borrar*). Debes indicar qué elemento deseas eliminar inmediatamente después de la palabra **del**:

```
>>> a = [1, 2, 3] ↲
```




```
>>> del a[1] ↲
```




```
>>> a ↲
[1, 3]
```

La sentencia **del** *no* produce una copia de la lista sin la celda borrada, sino que modifica directamente la lista sobre la que opera. Fíjate en qué efecto produce si dos variables apuntan a la misma lista:

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

Las cadenas son inmutables (y III)

Recuerda que las cadenas son inmutables. Esta propiedad también afecta a la posibilidad de borrar elementos de una cadena:

```
>>> a = 'Hola' ↲
>>> del a[1] ↲
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item deletion
```

El borrado de elementos de una lista es peligroso cuando se mezcla con el recorrido de las mismas. Veámoslo con un ejemplo. Hagamos un programa que elimina los elementos negativos de una lista.

solo_positivos_5.py — solo_positivos.py

```
1 a = [1, 2, -1, -4, 5, -2]
2
3 for i in a:
4     if i < 0:
5         del i
6
7 print a
```

¡Mal! Estamos usando **del** sobre un escalar (i), no sobre un elemento indexado de la lista (que, en todo caso, sería `a[i]`). Este es un error típico de principiante. La sentencia **del** no se usa así. Vamos con otra versión:

solo_positivos_6.py — solo_positivos.py

```
1 a = [1, 2, -1, -4, 5, -2]
2
3 for i in range(0, len(a)):
4     if a[i] < 0:
5         del a[i]
6
7 print a
```

Ahora sí usamos correctamente la sentencia **del**, pero hay otro problema. Ejecutemos el programa:

```
Traceback (most recent call last):
  File "solo_positivos.py", line 4, in ?
    if a[i] < 0:
IndexError: list index out of range
```

El mensaje de error nos dice que tratamos de acceder a un elemento con índice fuera del rango de índices válidos. ¿Cómo es posible, si la lista tiene 6 elementos y el índice i toma valores desde 0 hasta 5? Al eliminar el tercer elemento (que es negativo), la lista ha pasado a tener 5 elementos, es decir, el índice de su último elemento es 4. Pero el bucle «decidió» el rango de índices a recorrer antes de borrarse ese elemento, es decir, cuando la lista tenía el valor 5 como índice del último elemento. Cuando tratamos de acceder a `a[5]`, Python detecta que estamos fuera del rango válido. Es necesario que el bucle «actualice» el valor del último índice válido con cada iteración:

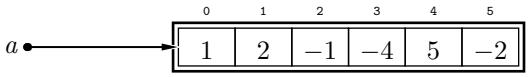
solo_positivos_7.py — solo_positivos.py

```
a = [1, 2, -1, -4, 5, -2]

i = 0
while i < len(a):
    if a[i] < 0:
        del a[i]
    i += 1

print a
```

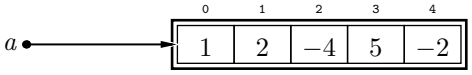
Ejecutemos el programa:

```
[1, 2, -4, 5]
```

¡No ha funcionado! El -4 no ha sido eliminado. ¿Por qué? Inicialmente la lista era:



Al eliminar el elemento a[2] de la lista original, i valía 2.



Después del borrado, incrementamos i y eso hizo que la siguiente iteración considerara el posible borrado de a[3], pero en ese instante -4 estaba en a[2] (fíjate en la última figura), así que nos lo «saltamos». La solución es sencilla: sólo hemos de incrementar i en las iteraciones que no producen borrado alguno:

solo_positivos_8.py — solo_positivos.py

```
a = [1, 2, -1, -4, 5, -2]

i = 0
while i < len(a):
    if a[i] < 0:
        del a[i]
    else:
        i += 1

print a
```

Ejecutemos el programa:

```
[1, 2, 5]
```

¡Ahora sí!

EJERCICIOS

▶ 232 ¿Qué sale por pantalla al ejecutar este programa?:

```
a = range(0, 5)
del a[1]
del a[1]
print a
```

▶ 233 Diseña un programa que elimine de una lista todos los elementos de *índice* par y muestre por pantalla el resultado.

(Ejemplo: si trabaja con la lista [1, 2, 1, 5, 0, 3], ésta pasará a ser [2, 5, 3].)

▶ 234 Diseña un programa que elimine de una lista todos los elementos de *valor* par y muestre por pantalla el resultado.

(Ejemplo: si trabaja con la lista `[1, -2, 1, -5, 0, 3]`, ésta pasará a ser `[1, 1, -5, 3]`.)

▶ 235 A nuestro programador novato se le ha ocurrido esta otra forma de eliminar el elemento de índice i de una lista a:

```
1 a = a[:i] + a[i+1:]
```

¿Funciona? Si no es así, ¿por qué? Y si funciona correctamente, ¿qué diferencia hay con respecto a usar **del** $a[i]$?

. . . . . .

La sentencia **del** también funciona sobre cortes:

```
>>> a = [1, 2, 3, 4, 5, 6] ↲
>>> del a[2:4] ↲
>>> a ↲
[1, 2, 5, 6]
```