Variables (2)

En esta lección se trata la relación entre variables y objetos en Python, haciendo hincapié en la modificación de variables.

Variables y objetos

Para entender cómo funcionan las variables en Python, hace falta tener en cuenta tres aspectos:

- Los valores (números, cadenas, listas, etc.) se almacenan en objetos.
- Hay dos tipos de objetos:
  - objetos inmutables (que no pueden cambiar su valor) como números, cadenas o tuplas,
  - objetos mutables (que pueden cambiar) como listas y diccionarios.
- El acceso a los valores almacenados en los objetos se realiza mediante variables, asignando las variables a los objetos.
- Las variables son simples etiquetas para hacer referencia a los objetos y son independientes de ellos.

Creación de variables

Cuando se asigna por primera vez un valor a una variable, Python:

- crea un objeto para almacenar el valor (si no existe todavía),
- crea la variable
- asocia la variable al objeto.

El valor de la variable es el valor almacenado por el objeto.

Modificación de variables

Cuando se asigna un nuevo valor a una variable creada anteriormente, el valor de la variable será el nuevo valor almacenado, pero con respecto a los objetos (que son los que realmente almacenan los valores) pueden darse tres situaciones:

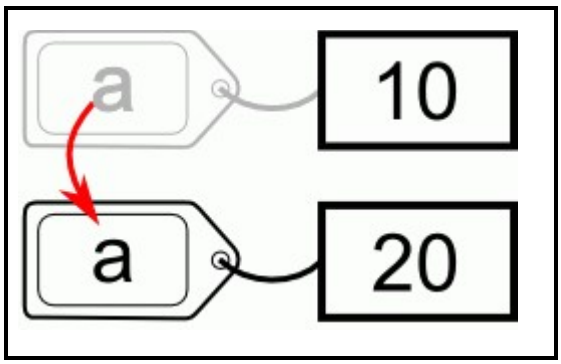
- que el objeto original sea inmutable y no pueda almacenar el nuevo valor. En este caso, la variable se asocia a un objeto distinto que contiene el nuevo valor.
- que el objeto original sea mutable, pero que no se modifique. En este caso, la variable se asocia a un objeto distinto que contiene el nuevo valor.
- que el objeto original sea mutable y pase a contener el nuevo valor. En este caso, la variable puede seguir asociada al mismo objeto.

Los tres ejemplos siguientes, ilustran estas tres situaciones. Los ejemplos están acompañados de dibujos en los que se muestran los objetos (con los valores que almacenan) y las variables (como etiquetas asociadas a los objetos).

- Objeto inmutable

Cambio de valor de una variable (objetos inmutables)

```
>>> a = 10
>>> a
10
>>> a = 20
>>> a
20
```

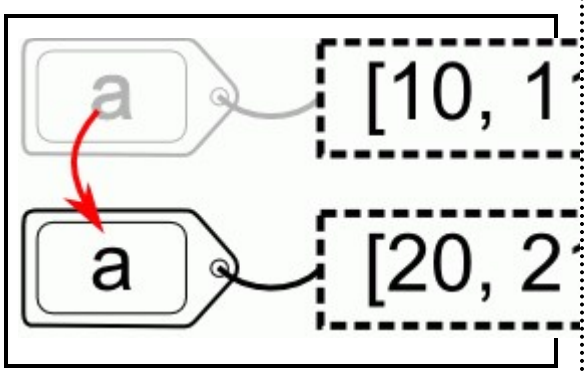


Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

- Objeto mutable que no se modifica

Cambio de valor de una variable (objetos mutables que no se modifican)

```
>>> a = [10, 11]
>>> a
[10, 11]
>>> a = [20, 21]
>>> a
[20, 21]
```

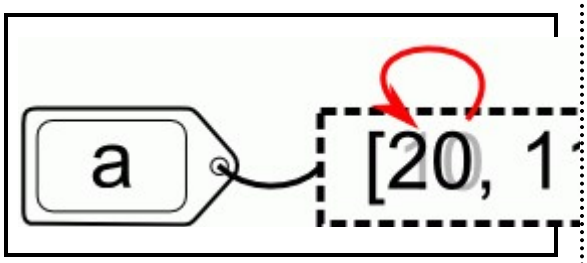


Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

- Objeto mutable que sí se modifica

Cambio de valor de una variable (objeto mutable que sí se modifica)

```
>>> a = [10, 11]
>>> a
[10, 11]
>>> a[0] = 20
>>> a
[20, 11]
```



Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

Cómo afecta la modificación de una variable a otras variables

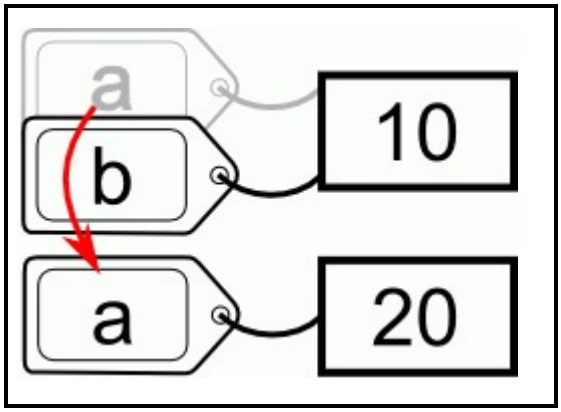
En Python, al modificar una variable se pueden modificar otras variables. Eso ocurre cuando al modificar una variable se modifique el objeto y haya otras variables haciendo referencia al mismo objeto

Para entenderlo, vamos a repetir los tres ejemplos anteriores introduciendo una segunda variable y observando qué ocurre en cada caso.

- Objeto inmutable

Cambio de valor de una variable (objetos inmutables)

```
>>> a = 10
>>> b = a
>>> a, b
(10, 10)
>>> a = 20
>>> a, b
(20, 10)
```

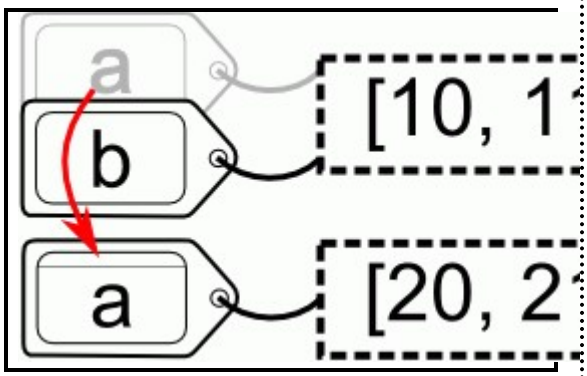


Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

- Objeto mutable que no se modifica

Cambio de valor de una variable (objetos mutables que no se modifican)

```
>>> a = [10, 11]
>>> b = a
>>> a, b
([10, 11], [10, 11])
>>> a = [20, 21]
>>> a, b
([20, 21], [10, 11])
```

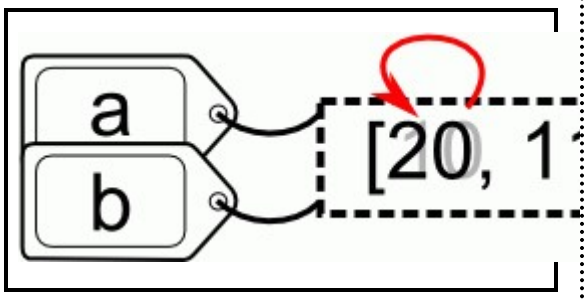


Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

- Objeto mutable que sí se modifica

Cambio de valor de una variable (objeto mutable que sí se modifica)

```
>>> a = [10, 11]
>>> b = a
>>> a, b
([10, 11], [10, 11])
>>> a[0] = 20
>>> a, b
([20, 11], [20, 11])
```

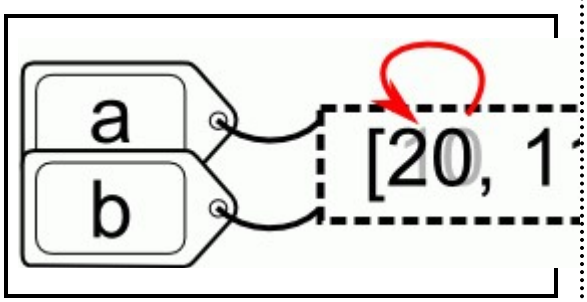


Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

En este último caso, si al cambiar la variable se cambia el objeto, da lo mismo qué variable cambie el objeto:

Cambio de valor de una variable (objeto mutable que sí se modifica)

```
>>> a = [10, 11]
>>> b = a
>>> a, b
([10, 11], [10, 11])
>>> b[0] = 20
>>> a, b
([20, 11], [20, 11])
```



Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

Modificación de variables asociadas a listas

Los objetos inmutables no se pueden modificar de ninguna manera. Cuando una variable está asociada a un objeto inmutable, al modificar la variable no se modifica el objeto, sino que la variable se asocia a un nuevo objeto.

Sin embargo, los objetos mutables se pueden modificar. Cuando una variable está asociada a un objeto mutable, al modificar la variable se puede modificar el objeto en algunos casos, pero no en otros.

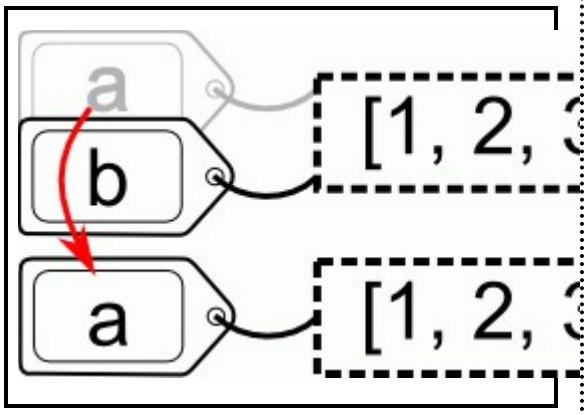
Veamos en el caso de las listas las diferentes maneras de modificar la variable y en qué casos no modifica el objeto y en qué casos sí.

Modificación de variables que no modifican las listas

- El operador **+** no modifica el objeto, como muestra el ejemplo siguiente:

Cambio de valor de una variable (a = a + [4])

```
>>> a = [1, 2, 3]
>>> b = a
>>> a, b
([1, 2, 3], [1, 2, 3])
>>> a = a + [4]
>>> a, b
([1, 2, 3, 4], [1, 2, 3])
```



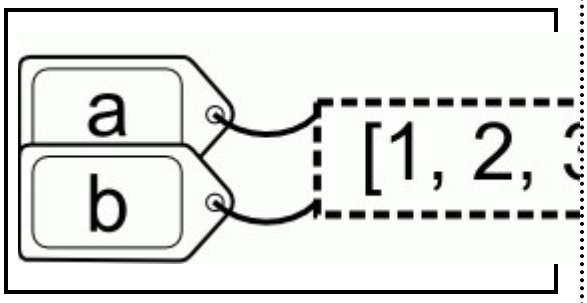
Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

Modificación de variables que modifican las listas

- El operador **+=** sí que modifica el objeto, como muestra el ejemplo siguiente:

Cambio de valor de una variable (a += [4])

```
>>> a = [1, 2, 3]
>>> b = a
>>> a, b
([1, 2, 3], [1, 2, 3])
>>> a += [4]
>>> a, b
([1, 2, 3, 4], [1, 2, 3, 4])
```

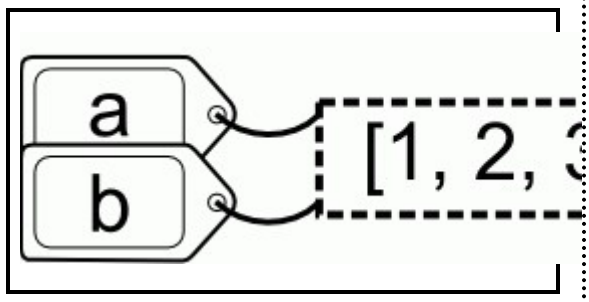


Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

- El método **append()** sí que modifica el objeto, como muestra el ejemplo siguiente:

Cambio de valor de una variable (a.append(4))

```
>>> a = [1, 2, 3]
>>> b = a
>>> a, b
([1, 2, 3], [1, 2, 3])
>>> a.append(4)
>>> a, b
([1, 2, 3, 4], [1, 2, 3, 4])
```

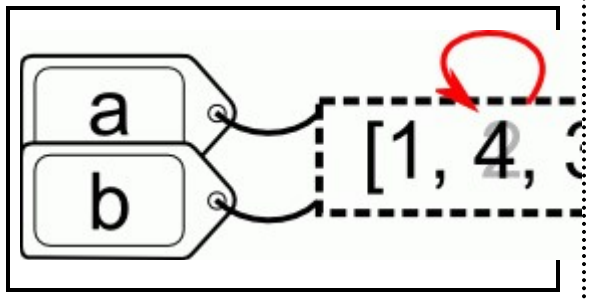


Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

- Modificando un valor de una lista sí que modifica el objeto, como muestra el ejemplo siguiente:

Cambio de valor de una variable (a[1] = 4)

```
>>> a = [1, 2, 3]
>>> b = a
>>> a, b
([1, 2, 3], [1, 2, 3])
>>> a[1] = 4
>>> a, b
([1, 4, 3], [1, 4, 3])
```



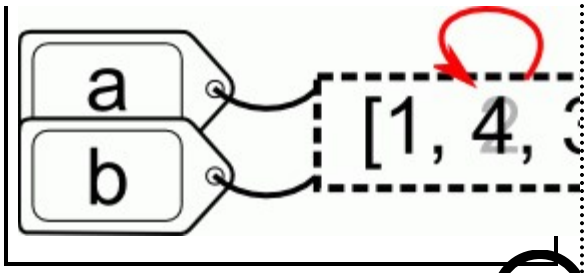
Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

- Modificando un valor de una sublista sí que modifica el objeto, como muestra el ejemplo siguiente:

Cambio de valor de una variable (a[1:2] = [4])

```
>>> a = [1, 2, 3]
>>> b = a
>>> a, b
([1, 2, 3], [1, 2, 3])
>>> a[1:2] = [4]
```

```
>>> a, b
([1, 4, 3], [1, 4, 3])
```



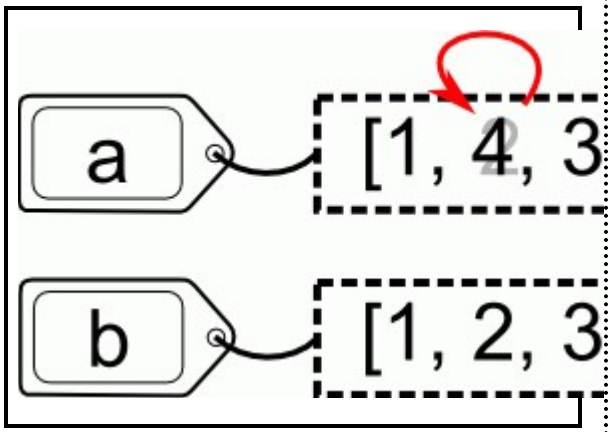
Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

Copiar una lista

Como se comenta en la lección de Listas, para copiar una lista hay que utilizar la notación de sublistas. De esta manera se consiguen tener dos objetos independientes, como muestra el ejemplo siguiente

Copiar una lista

```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> a, b
([1, 2, 3], [1, 2, 3])
>>> a[1] = [4]
>>> a, b
([1, 4, 3], [1, 2, 3])
```



Puede ver la ejecución paso a paso de este ejemplo utilizando los iconos de avance y retroceso situados abajo a la derecha.

Última modificación de esta página: 22 de marzo de 2015