Introducción a las listas

Python nos permite definir secuencias de valores de cualquier tipo. Por ejemplo, secuencias de números o de símbolos. A esto se le llama *lista*. La lista viene determinada por una serie de valores encerrados entre corchetes y separados por comas.

In [1]:
```
a =  [2, 3, 4, 7]
```

Los elementos de una lista pueden ser cadenas de caracteres o incluso expresiones que se calculan en tiempo real

In [2]:
```
b = ['hola', 'adios']
```

In [3]:
```
c = [1, 1+3, 6/2]
```

In [4]:
```
c
```

Out[4]:
```
[1, 4, 3]
```

Una lista puede no tener elementos: *la lista vacía*

In [5]:
```
void = []
```

In [6]:
```
void
```

Out[6]:
```
[]
```

Operaciones básicas

Longitud: podemos calcular la longitud de una cadena con la orden *len*

In [7]:
```
len(a)
```

Out[7]:
```
4
```

In [8]:
```
len(void)
```

Out[8]:
```
0
```

Concatenación: podemos concatenar cadenas con el operador +

In [9]:
```
a = a + b
```

In [10]:
```
a
```

Out[10]:
```
[2, 3, 4, 7, 'hola', 'adios']
```

Repetición: podemos repetir cadenas con el operador *

In [11]:
```
e = a*2
```

In [12]:
```
e
```

Out[12]:
```
[2, 3, 4, 7, 'hola', 'adios', 2, 3, 4, 7, 'hola', 'adios']
```

Podemos acceder a cada elemento de la lista a través de su índice. *Recuerda que los índices siempre empiezan a contar en 0*

In [13]:
```
a = [23, 45, 67]
a
```

Out[13]:
```
[23, 45, 67]
```

In [14]:
```
a[0]
```

Out[14]:
```
23
```

In [15]:
```
a[3]# cuidado con salirnos del rango
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-15-f8a037bae13f> in <module>()
----> 1 a[3]# cuidado con salirnos del rango

IndexError: list index out of range
```

In [16]:
```
f = [a[0], a[2]]
```

In [17]:
```
f
```

Out[17]:
```
[23, 67]
```

Los índices pueden ser *negativos*.

In [18]:
```
z = range(4,10)
z
```

Out[18]:
```
[4, 5, 6, 7, 8, 9]
```

In [19]:
```
z[-1]
```

Out[19]:
```
9
```

In [20]:
```
z[-6]
```

Out[20]: 4

Operador de corte: Podemos quedarnos con un trozo de una lista

In [21]:
```
a = [1, 2, 3, 45, 66]
```

In [22]:
```
b = a[1:4]
b
```

Out[22]: [2, 3, 45]

In [23]:
```
b = a[1:]
b
```

Out[23]: [2, 3, 45, 66]

In [24]:
```
b = a[:4]
b
```

Out[24]: [1, 2, 3, 45]

In [25]:
```
b = a[:]
b
```

Out[25]: [1, 2, 3, 45, 66]

Podemos utilizar *índices negativos*

In [26]:
```
c = a[-5:-1]
c
```

Out[26]: [1, 2, 3, 45]

In [27]:
```
#qué significa esto
print a, b, b == a
```

```
[1, 2, 3, 45, 66] [1, 2, 3, 45, 66] True
```

In [29]:
```
b is a
```

Out[29]: False

El bucle **for-in** recorre los elementos de una lista.

In [30]:
```
s = ''
for i in a:
    s = s + str(i)
s
```

Out[30]: '1234566'

range es un caso particular de lista

In [31]:
```
range(15)
```

Out[31]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

In [32]:
```
a = a + range(8)
```

In [33]:
```
a
```

Out[33]:
```
[1, 2, 3, 45, 66, 0, 1, 2, 3, 4, 5, 6, 7]
```

In [34]:
```
s = ''
for n in a:
    s = s + str(n)
s
```

Out[34]:
```
'123456601234567'
```

Comparación de listas Las listas se pueden comparar con los operaciones habituales: ==, !=, <=, ...

In [35]:
```
a = [3, 2, 3, 99, 4]
b = [3, 4]
a == b
```

Out[35]:
```
False
```

In [36]:
```
a != b
```

Out[36]:
```
True
```

In [37]:
```
a < b
```

Out[37]:
```
True
```

In [38]:
```
c = a
c
```

Out[38]:
```
[3, 2, 3, 99, 4]
```

In [39]:
```
a == c
```

Out[39]:
```
True
```

In [40]:
```
d = a[:]
d
```

Out[40]:
```
[3, 2, 3, 99, 4]
```

In [41]:
```
c == d
```

Out[41]:
```
True
```

In [42]:
```
c is d
```

Out[42]:
```
False
```

Modificación de elementos de una lista. Mutabilidad. El operador is

In [43]:
```
#Dada una lista, podemos modificar sus componentes
a = range(4,10)
a
```

Out[43]:
```
[4, 5, 6, 7, 8, 9]
```

In [44]:
```
a[2] = 77
a
```

Out[44]:
```
[4, 5, 77, 7, 8, 9]
```

In [45]:
```
'''
EJEMPLO: Elimina los elementos negativos de una lista,
sustituyéndolos por 0
'''
b = [1, 3, -4, 5, -2, 3, -9, -100, 77]
for i in range(len(b)):
    if b[i] < 0:
        b[i] = 0
b
```

Out[45]:
```
[1, 3, 0, 5, 0, 3, 0, 0, 77]
```

Mutabilidad, inmutabilidad --> de qué me estás hablando?

In [46]:
```
a = range(4,10)
a
```

Out[46]:
```
[4, 5, 6, 7, 8, 9]
```

In [47]:
```
b = a
b
```

Out[47]:
```
[4, 5, 6, 7, 8, 9]
```

In [48]:
```
a[3] = 99
a
```

Out[48]:
```
[4, 5, 6, 99, 8, 9]
```

In [49]:
```
b
```

```
In [49]: b
         a is b

Out[49]: True
```

Esto es un poco raro, no?

```
In [59]: c = a[:]
         c

Out[59]: [4, 5, 6, 99, 8, 9]

In [60]: c[3] = 44
         c

Out[60]: [4, 5, 6, 44, 8, 9]

In [61]: a

Out[61]: [4, 5, 6, 99, 8, 9]

In [62]: b

Out[62]: [4, 5, 6, 99, 8, 9]
```

¿Por qué pasa esto? a y b son esencialmente los mismo (señalan a las mismas celdas de memoria). a y c son distintos, la información están guardada en distintas celdas de memoria.

```
In [63]: c = a
         b = a[:]

In [64]: print a == c, a == b

         True True

In [65]: print a is c, a is b ##

         True False
```

Las cadenas de caracteres se comportan de forma parecida, pero no igual

```
In [66]: s = 'hola'
         s[0]

Out[66]: 'h'

In [67]: s[0] = 'j'

         ---------------------------------------------------------------------
         --
         TypeError                                 Traceback (most recent call las
         t)
```

```
/home/jesus/Dropbox/docencia13-14/ipython/<ipython-input-67-a225757de94d>
 in <module>()
----> 1 s[0] = 'j'

TypeError: 'str' object does not support item assignment
```

Añadir elementos a una lista

Podemos añadir elementos a una lista de dos formas diferentes. Una manera es utilizando el operador de concatenación. Esto crea una lista nueva con el nuevo elemento.

In [16]:
```
a = range(4,10)
b = a
```

In [17]:
```
a = a + [99]
print a, b
```

```
[4, 5, 6, 7, 8, 9, 99] [4, 5, 6, 7, 8, 9]
```

In [18]:
```
print a == b, a is b
```

```
False False
```

También podemos añadir elementos con el método *append*. De esta forma se modifica la lista original, sin crear una lista nueva.

In [4]:
```
c = b
c
```

Out[4]:
```
[4, 5, 6, 7, 8, 9]
```

In [5]:
```
c.append(99) #ATENTOS A LA MANERA DE UTILIZAR append
print c, b
```

```
[4, 5, 6, 7, 8, 9, 99] [4, 5, 6, 7, 8, 9, 99]
```

In [6]:
```
print c == b, c is b
```

```
True True
```

EJEMPLO: Crea una lista con los cuadrados de los 10 primeros números.

In [20]:
```
L = []
i = 1
while i <= 15:
    L.append(i*i)
    i+=1
L
```

Out[20]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225]
```

Ejemplos

Recorridos de listas. Utilización en funciones

In [21]:
```
'''
Si a es una lista de números, escribe un programa
que modifique la lista de forma que cada
componente sea igual al cuadrado del componente original.
'''

def cuadrado_lista (a):
    '''
    Eleva al cuadrado los elementos de una lista
    '''
    for i in range(len(a)):
        a[i] = a[i]**2

#No hace falta poner return: modifico la propia lista
```

In [22]:
```
lista = range(10)+range(3)
lista
```

Out[22]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2]
```

In [23]:
```
cuadrado_lista(lista)
lista
```

Out[23]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 0, 1, 4]
```

In [24]:
```
#podemos representar mejor la lista
def dibuja(lista):
    # presenta por pantalla una lista
    k = len(lista)
    for i in range(k-1):
        print lista[i], '*',
    print lista[k-1]
```

In [25]:
```
dibuja(lista)
```

```
0 * 1 * 4 * 9 * 16 * 25 * 36 * 49 * 64 * 81 * 0 * 1 * 4
```

In [26]:
```
# creación de una lista aleatoria
def random_list(size):
    #Creo una lista aleatoria
    from random import randint
    # randint(a, b) devuelve un entero aleatorio N,
    #a <= N <= b
    data = []
```

```
    for i in range(size):
        data.append(randint(-10,10))
    return data
```

In [29]:
```
milista = random_list(10)
dibuja(milista)
```

```
-3 * -2 * -6 * 3 * -4 * 5 * 2 * 9 * -8 * -8
```

In [15]:
```
cuadrado_lista(milista)
dibuja(milista)
```

```
9 , 49 , 36 , 1 , 0 , 1 , 16 , 49 , 4 , 25 , 25 , 0 , 64 , 49 , 100
```

Ejemplo: Buscar en una lista --> ver buscar.py

In []:
```

```