**Tuplas:**

tup = tuple(‘string’)

tup

(‘s’,’t’,’r’,’i’,’g’)

Si un objeto dentro de una tupla es mutable (ej.lista) se puede modificar.

tup = tuple(['foo', [1, 2], True])

tup[1].append(3)

(['foo', [1, 2,3], True])

se pueden concatenar tuplas con el operador +

(4, None, 'foo') + (6, 0) + ('bar',)

Out[14]: (4, None, 'foo', 6, 0, 'bar')

se pueden multiplicar por un entero como las listas

('foo', 'bar') \* 4

Out[15]: ('foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'bar')

Se puede asignar una tupla como expresión de variables

tup = (4, 5, 6)

a, b, c = tup

b # la variable b tiene el valor de 5

5

Sintaxis \*rest se utiliza para extraer una lista de elementos de una lista

values = 1, 2, 3, 4, 5

a, b, \*rest = values

a, b

(1, 2)

rest

[3, 4, 5]

As a matter of convention, many Python programmers will use the underscore (\_) for unwanted variables:

a, b, \*\_ = values

**Listas**

Añadir elementos a la lista

b\_list.append('dwarf')

b\_list

['foo', 'peekaboo', 'baz', 'dwarf']

Using insert you can insert an element at a specific location in the list:

b\_list.insert(1, 'red')

b\_list

['foo', 'red', 'peekaboo', 'baz', 'dwarf']

The inverse operation to insert is pop, which removes and returns an element at a particular index:

In [50]: b\_list.pop(2)

Out[50]: 'peekaboo'

In [51]: b\_list

Out[51]: ['foo', 'red', 'baz', 'dwarf']

Elements can be removed by value with remove, which locates the first such value and removes it from the list:

In [52]: b\_list.append('foo')

In [53]: b\_list

Out[53]: ['foo', 'red', 'baz', 'dwarf', 'foo']

In [54]: b\_list.remove('foo')

In [55]: b\_list

Out[55]: ['red', 'baz', 'dwarf', 'foo']

Check if a list contains a value using the in keyword:

dwarf' **in** b\_list

True

The keyword not can be used to negate in:

'dwarf' **not** **in** b\_list

False

Se pueden concatenar dos listas con el opeador +

If you have a list already defined, you can append multiple elements to it using the extend method:

x = [4, None, 'foo']

x.extend([7, 8, (2, 3)])

x

[4, None, 'foo', 7, 8, (2, 3)]

is usually preferable. Thus,

everything = []

**for** chunk **in** list\_of\_lists:

everything.extend(chunk)

is faster than the concatenative alternative:

everything = []

**for** chunk **in** list\_of\_lists:

everything = everything + chunk

la función sort ordena los valores de una lista de menor a mayor

a = [7, 2, 5, 1, 3]

a.sort()

a

[1, 2, 3, 5, 7]

a function that produces a value to use to sort the objects. For example, we could sort a collection of strings by their lengths:

b = ['saw', 'small', 'He', 'foxes', 'six']

b.sort(key=len)

b

['He', 'saw', 'six', 'small', 'foxes']

bisect module implements binary search and insertion into a sorted list. bisect.bisect finds the location where an element should be inserted to keep it sorted, while bisect.insort actually inserts the element into that location (es necesario que la lista esté previamente ordenada para utilizar esta función correctamente):

**import** **bisect**

c = [1, 2, 2, 2, 3, 4, 7]

bisect.bisect(c, 2)

4

bisect.bisect(c, 5)

6

bisect.insort(c, 6)

c

[1, 2, 2, 2, 3, 4, 6, 7]

Slicing