Bucle `for_in`

Sintaxis:

```
for <<variable>> in <<secuencia>>:      <<cuerpo del bucle>>
```

Bucles `for_in` con listas

In [38]:
```
def assessment(grade_list):
    """
    Computes the average of a list of grades

    @type grades: [float]
    @rtype: float
    """
    average = 0.0
    for grade in grade_list:
        average += grade
    return average/len(grade_list)
```

In [39]:
```
assessment([4.5, 6, 5]), assessment([4, 6, 5, 7, 5, 6, 8])
```

Out[39]: (5.166666666666667, 5.857142857142857)

In [40]:
```
def short_names(name_list, n):
    """
    From a list of names, the function chooses the names with length below of equal n.
    All the names that satisfy the condition are returned in a list.

    @type name_list: [string]
    @type n: int
    @rtype: [string]
    """
    short = []
    for name in name_list:
        if len(name) <= n:
            short.append(name)
    return short
```

In [41]:
```
l = ['Ana', 'Marta', 'Patricia', 'Alba', 'Silvia', 'Gloria', 'Lara']
short_names(l, 5), short_names(l, 3)
```

Out[41]: (['Ana', 'Marta', 'Alba', 'Lara'], ['Ana'])

La funciones `range()` y `xrange()`

La función `range()` genera listas de forma muy versatil. Una manera muy frecuente de hacer bucles `for_in` es generando la lista que hace de secuencia con la función `range()`. Veamos algunos detalles sobre esta función `range()`.

In [42]:
```
xrange(10)
```

Out[42]: xrange(10)

In [43]:
```
range(3, 12)
```

Out[43]: [3, 4, 5, 6, 7, 8, 9, 10, 11]

In [44]:
```
range(5, 60, 5)
```

Out[44]: [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55]

In [45]:
```
range(10, 2)
```

Out[45]: []

In [46]:
```
range(10, 2, -1)
```

Out[46]: [10, 9, 8, 7, 6, 5, 4, 3]

In [47]:
```
a=6
b=10
range(a-1, (b*2)-3)
```

Out[47]: [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

La función `xrange()` es parecida a la función range. Es perezosa, no genera la lisa hasta que alguien le pide que la genere, más útiles en relación con bucles for. Nota, en Python 3 la función `range` se comporta como la `xrange` de Python 2.

In [48]:
```
xrange(10)
```

Out[48]: xrange(10)

In [49]:
```
list(xrange(10))
```

Out[49]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Bucles `for_in` para listas generadas con `range()`

In [50]:
```
def random_list(n):
    """
    Returns a list of n random integer between 0 and 100.

    @type n: int
    @rtype: [int]
    """
    import random
    result = []
    for x in xrange(n):
        result.append(random.randint(0,100))
    return result
```

In [51]:
```
random_list(3), random_list(5)
```

Out[51]: ([30, 37, 25], [35, 21, 39, 11, 65])

In [52]:
```
def random_list(n, minimum, maximum):
    """
    Returns a list of n random integer between minimum and maximum.

    @type n: int
    @type minimum: int
    @type maximum: int
    @rtype: [int]
    """
    import random
    result = []
    for x in xrange(n):
        result.append(random.randint(minimum,maximum))
    return result
```

In [53]:
```
random_list(3,1,5), random_list(5,0,1000)
```

Out[53]: ([4, 2, 4], [695, 450, 42, 742, 721])

In [54]:
```
def multiple_7_and_5(n):
    """
    Returns the list of numbers below n that are, at the same time,
    multiple of 7 and 5.

    @type n: int
    @rtype: [int]
    """
    result = []
    for x in xrange(n):
```

```
        if x%5 == 0 and x%7 == 0 :
            result.append(x)
    return result
```

In [55]:
```
multiple_7_and_5(100)
```

Out[55]: [0, 35, 70]

Si no nos gusta que aparezca el 0, podemos hacer que el rango comience en 1.

In [56]:
```
def multiple_7_and_5(n):
    """
    Returns the list of numbers below n that are, at the same time,
    multiple of 7 and 5.

    @type n: int
    @rtype: [int]
    """
    result = []
    for x in xrange(1, n):
        if x%5 == 0 and x%7 == 0 :
            result.append(x)
    return result
```

In [57]:
```
multiple_7_and_5(100)
```

Out[57]: [35, 70]

Pero... no es una forma muy eficaz, hay muchos números de los que podemos fácilmente *'librarnos'*

In [58]:
```
def multiple_7_and_5(n):
    """
    Returns the list of numbers below n that are, at the same time,
    multiple of 7 and 5.

    @type n: int
    @rtype: [int]
    """
    result = []
    for x in xrange(7, n, 7):
        if x%5 == 0:
            result.append(x)
    return result
```

In [59]:
```
multiple_7_and_5(100)
```

Out[59]: [35, 70]

In [60]:
```
def reverse(initial_list):
    """
    Returns a list with the elements of initial_list reversed, that is, the first element of
    initial_list would be the last element, the second element of initial_list woulb be the
    second to last...

    @type initial_list: list
    @rtype: list
    """
    result = []
    start = len(initial_list)-1
    end = -1
    for i in xrange(start, end, -1):
        result.append(initial_list[i])
    return result
```

In [61]:
```
reverse([1,2,3,4]), reverse(["hola","buenas","tardes"])
```

Out[61]: ([4, 3, 2, 1], ['tardes', 'buenas', 'hola'])

Bucles `for_in` en strings

In [76]:
```
for c in 'hola':
    print c
```

```
h
o
l
a
```

In [77]:
```
for c in 'buenas tardes':
    print c.lower(), ord(c.lower()), c.upper(), ord(c.upper())
```

```
b 98 B 66
u 117 U 85
e 101 E 69
n 110 N 78
a 97 A 65
s 115 S 83
  32   32
t 116 T 84
a 97 A 65
r 114 R 82
d 100 D 68
e 101 E 69
s 115 S 83
```

In [64]:
```
def letter_count(letter, word):
    """
    Counts the occurrences of letter in word.

    @type letter: string
    @type word: string
    @rtype: int
    """
    cont = 0
    for char in word:
        if char == letter:
            cont = cont + 1
    return cont
```

In [65]:
```
letter_count('o', 'pelirrojo')
```

Out[65]: 2

In [66]:
```
letter_count('j', 'pelirrojo')
```

Out[66]: 1

In [67]:
```
letter_count('a', 'pelirrojo')
```

Out[67]: 0

In [68]:
```
letter_count('J', 'pelirrojo')
```

Out[68]: 0

In [69]:
```
def letter_count(letter, word):
    """
    Counts the occurrences of letter in word. This function is not case sensitive, that
    is letter_count('A', 'Ana') = 2 and letter_count('a', 'ANA') = 2.

    @type letter: string
    @type word: string
    @rtype: int
    """
    cont = 0
```

```
    for char in word:
        if char.upper() == letter.upper():
            cont = cont + 1
    return cont
```

In [70]:
```
letter_count('j', 'Pelirrojo')
```

Out[70]: 1

In [71]:
```
letter_count('J', 'Pelirrojo')
```

Out[71]: 1

In [72]:
```
letter_count('p', 'Pelirrojo')
```

Out[72]: 1

In [72]:
```

```