Bucles y Listas

Un poco de motivación

Queremos hacer la media de unos números reales, cada uno en una variable. ¿qué ocurre si no se sabe cuántos numeros hay?

In [36]:
```
numberlst = [4.0, 6.0, 7.0, 3.0, 2.0]
```

In [37]:
```
s = 0.0
s = s + numberlst[0]
s = s + numberlst[1]
s = s + numberlst[2]
s = s + numberlst[3]
s = s + numberlst[4]
mean = s / len(numberlst)
mean
```

Out[37]: 4.4

Eso no es generalizable. Necesitamos un bucle

In [38]:
```
def mean(numberlst):
    """This function returns the mean of a list of numbers
    @type numberlst: [float]
    @rtype float
    """
    s = 0.0
    i = 0
    while i<len(numberlst):
        s = s + numberlst[i]
        i = i + 1
    return s / len(numberlst)
mean(numberlst)
```

Out[38]: 4.4

Queremos hacer funciones con polinomios

poly ----> $3x^2 - 6x + 2$

Como representamos este polinomio, ¿con 4 reales? ¿y si el polinomio tiene grado n? Necesitamos algo que nos permita representar n reales: 3, 0, -6, 3

In [39]:
```
poly = [2.0, -6.0, 0, 3.0]
poly[0], poly[1], poly[2], poly[3]
```

Out[39]: (2.0, -6.0, 0, 3.0)

Trabajar con listas

La longitud de una lista

In [40]:
```
len(poly)
```

Out[40]:
```
4
```

Las posiciones de la lista poly van desde 0 hasta len(poly). Ojo a los accesos ilegales

In [41]:
```
poly[4]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-41-a085d43b506f> in <module>()
----> 1 poly[4]

IndexError: list index out of range
```

Se puede cambiar el contenido de una posición de una lista

In [42]:
```
poly[3] = -7.0
poly
```

Out[42]:
```
[2.0, -6.0, 0, -7.0]
```

In [43]:
```
poly[4] = 9.0
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-43-d742d21044b4> in <module>()
----> 1 poly[4] = 9.0

IndexError: list assignment index out of range
```

Los elementos de una lista se pueden usar en cualquier contexto

In [44]:
```
a = poly[2] + 1 # expresión
abs(poly[3])    #llamada a función
```

Out[44]:
```
7.0
```

Se pueden añadir más elementos a una lista

In [45]:
```
poly.append(9.0)
poly
```

Out[45]:
```
[2.0, -6.0, 0, -7.0, 9.0]
```

Ejemplos de bucles con listas

Vamos a hacer una función que evalúe el polinomio en el punto x

In [46]:
```
x = 2
result = 0.0
power = 1
result = result + power * poly[0]
power = power * x
result = result + power * poly[1]
power = power * x
result = result + power * poly[2]
power = power * x
result = result + power * poly[3]
result
```

Out[46]:
```
-66.0
```

¿y si el polinomio tiene grado n?

In [47]:
```
x = 2
result = 0.0
power = 1
degree = 4
i = 0
while i<=degree:
    result = result + power * poly[i]
    power = power * x
    i = i + 1
result
```

Out[47]:
```
78.0
```

In [48]:
```
def eval_poly(poly, x):
    """This function evaluates the polynomial poly at point x.
    Poly is a list of floats containing the coeficients of the polynomial
    poly[i] -> coeficient of degree i

    @type poly: [float]
    @type x: float
    @rtype float
    """
    result = 0.0
    power = 1
    degree = len(poly)-1
    i = 0
    while i<=degree:
        result = result + poly[i] * power
        power = power * x
        i = i + 1
    return result

eval_poly(poly, 2.0)
```

Out[48]:
```
78.0
```

Ejercicio propuesto: Descomposición en factores primos de un número.

Ejemplos:

60 = 2, 2, 3, 5

15 = 3, 5

Seguimos el algoritmo 'clásico' de descomposición, se comienza dividiendo el número original entre el divisor más pequeño posible, se acumula el dividendo y se continua con el divisor.

```
60|2 30|2 15|3  5|5  1
```

In [49]:
```
def factors(n):
    """
    This function computes the list of factors of n
    @type n: int
    @param n: n>1
    @rtype: [int]
    """
    fct = 2 # 2 is the first prime number
    factors = [] #Esto es la lista vacía
    while n>1:
        if n % fct == 0: # if fct divides n, it is a prime number
            factors.append(fct)
            n = n//fct
        else:
            fct += 1
    return factors

factors(392000)
```

Out[49]:
```
[2, 2, 2, 2, 2, 2, 5, 5, 5, 7, 7]
```

Si queremos calcular la multiplicidad podemos hacer que devuelva una lista de tuplas

In [50]:
```
def factors_exp(n):
    """
    This function returns the list of factors of n,
    together with their exponents. The function returns
    a list of tuples, the first element of the tuple and the
    second one is the exponent.

    @type n: int
    @param n: n>1
    @rtype: [[int]]
    """
    fct = 2
    factors = []
    while n>1:
        if n % fct == 0: #if it is a divisor, we compute its multiplicity.
            exp = 1
            n = n // fct
            while n % fct ==0:
                exp += 1
                n = n // fct
            factors.append([fct,exp])
        # at this point n % fct != 0, so we increase fct in any case.
        fct += 1
    return factors

factors_exp(60)
```

Out[50]:
```
[[2, 2], [3, 1], [5, 1]]
```

In [50]:
```

```