Ampliación de funciones

Hasta ahora hemos utilizado numerosas funciones para implementar nuestros algoritmos

In [1]:
```
def letra_dni(num):
    '''
    Calcula la letra del DNI a partir de la codificación estándar
    '''
    tabla = 'TRWAGMYFPDXBNJZSQVHLCKE'
    posicion = num % 23
    return tabla[posicion]
```

In [2]:
```
def es_perfecto(num):
    '''
    Un número es perfecto si es igual a la suma de sus divisores.
    Por ejemplo: 6 es un número perfecto porque
    sus divisores propios son 1, 2 y 3; y 6 = 1 + 2 + 3.
    Los siguientes números perfectos son 28, 496 y 8128.
    '''
    sumatorio = 0
    for i in range(1, num):
        if num % i == 0:
            sumatorio += i
    return sumatorio == num
```

Todas responden al mismo esquema:

In [4]:
```
def nombre(parametros):
    operacion1
    operacion2
    operacion_n
    return resultado
```

Vamos a revisitar algunos usos de las funciones, observando algunos fenómenos interesantes.

Listas como parámetros

Como ya sabemos, las listas se pueden utilizar como parámetros de funciones.

In [5]:
```
def sumatorio(lista):
    '''
    Suma los elementos de una lista
    '''
    suma = 0
    for numero in lista:
        suma += numero
    return suma
```

Observa la distinta utilización de un lista en una función.

In [6]:
```
def duplica(lista):
    for i in range(len(lista)):
        lista[i] = 2*lista[i]
```

In [7]:
```
def duplica2(lista):
    result = []
    for x in lista:
        result.append(2*x)
    return result
```

En el primer caso, la lista pasada como parámetro se modifica, en el segundo no.

In [8]:
```
mylist = range(5)
duplica(mylist)
print mylist
```

```
[0, 2, 4, 6, 8]
```

In [9]:
```
mylist = range(5)
list2 = duplica2(mylist)
print mylist, list2
```

```
[0, 1, 2, 3, 4] [0, 2, 4, 6, 8]
```

El primer ejemplo no funciona con tipos simples

In [10]:
```
def duplica_numero(n):
    n = 2*n
```

In [11]:
```
n = 10
duplica_numero(n)
print n
```

```
10
```

Funciones que devuelven un valor vacío. El valor None

In [12]:
```
def maximo(lista):
    '''
    Calcula el máximo de una lista
    '''
    if len(lista) > 0:
        candidato = lista[0]
        for elemento in lista:
            if elemento > candidato:
                candidato = elemento
    return candidato
```

In [13]:
```
print maximo(range(20))
```

```
19
```

In [14]:
```
print maximo([])
```

```
---------------------------------------------------------------------
---------
UnboundLocalError                         Traceback (most recent c
all last)
<ipython-input-14-2abb6b3f3588> in <module>()
----> 1 print maximo([])

<ipython-input-12-eba895a7e209> in maximo(lista)
      8             if elemento > candidato:
      9                 candidato = elemento
---> 10     return candidato

UnboundLocalError: local variable 'candidato' referenced before as
signment
```

¿Cómo puedo gestionar la lista vacía, si no tiene un máximo?

In [15]:
```
def maximo(lista):
    '''
    Calcula el máximo de una lista
    '''
    if len(lista) > 0:
        candidato = lista[0]
        for elemento in lista:
            if elemento > candidato:
                candidato = elemento
    else:
        candidato = None
    return candidato
```

In [17]:
```
print maximo([])
```

```
None
```

El valor None tiene entidad propia en python

In [18]:
```
print duplica(range(5))
```

```
None
```

Se puede utilizar cuando no queremos devolver un valor para alguno de los casos estudiados

Funciones sin parámetros

A veces utilizamos funciones que no tienen parámetros

In [20]:
```
def lee_entero_positivo():
    '''
    Lee un número positivo.
    '''
    print 'Número positivo = ',
    numero = int(raw_input())
    while numero < 0:
        print 'Error: el número debe ser positivo.'
        print 'Número positivo = ',
        numero = int(raw_input())
    return numero
```

In [22]:
```
lee_entero_positivo()
```

```
Número positivo = -2
 Error: el número debe ser positivo.
Número positivo = -1
 Error: el número debe ser positivo.
Número positivo = 3
```

Out[22]: 3

Otro ejemplo de utilización son los menús de usuario

In [24]:
```
from math import pi
def diametro(radio):
    return 2*radio
def perimetro_circulo(radio):
  return  2 * pi * radio
def area_circulo(radio):
  return  pi * radio ** 2

def menu():
  print 'Escoge una opción: '
  print 'a) Calcular el diámetro.'
  print 'b) Calcular el perímetro.'
  print 'c) Calcular el área.'
  print 'd) Salir'
  opcion = raw_input('Teclea a, b, c o d y pulsa el retorno de carr
o: ')
  return opcion

radio = float(raw_input('Dame el radio de un círculo: '))
opcion = ''#ARTIFICIO PARA ENTRAR EN EL BUCLE
while opcion != 'd':
        opcion = menu()
        if opcion == 'a':
          print 'El diámetro es', diametro(radio)
        elif opcion == 'b':
          print 'El perímetro es', perimetro_circulo(radio)
        elif opcion == 'c':
          print 'El área es', area_circulo(radio)
        elif opcion != 'd' :
          print 'Sólo hay 4 opciones: a, b, c o d. Tú has tecleado'
, opcion
print 'Gracias por utilizar el programa'
```

```
Dame el radio de un círculo: 4
Escoge una opción:
a) Calcular el diámetro.
b) Calcular el perímetro.
c) Calcular el área.
d) Salir
Teclea a, b, c o d y pulsa el retorno de carro: c
El área es 50.2654824574
Escoge una opción:
a) Calcular el diámetro.
b) Calcular el perímetro.
c) Calcular el área.
d) Salir
Teclea a, b, c o d y pulsa el retorno de carro: d
Gracias por utilizar el programa
```

Funciones sin return

In [25]:
```
def dibuja_lista(lista, separador):
    '''
    Representa una lista, de manera elegante
    '''
    for i in range(len(lista) - 1):
        print lista[i], separador,
    print lista[len(lista) - 1]
```

In [26]:
```
mylist = range(6)
dibuja_lista(mylist, '**')
```

```
0 ** 1 ** 2 ** 3 ** 4 ** 5
```

In [27]:
```
def dibuja_cuadrado(size):
    '''
    Dibuja un cuadrado de número consecutivos, de lado size
    '''
    num = 1
    for i in range(size): # FILAS
        for j in range(size): # COLUMNAS
            print '%4d' % (num),
            num += 1
        print
```

In [28]:
```
dibuja_cuadrado(3)
```

```
   1    2    3
   4    5    6
   7    8    9
```

Funciones que devuelven varios valores

In [29]:
```
def max_min(lista):
    '''
    Calcula el máximo y el mínimo de una lista
    '''
    if len(lista) > 0:
        maxi = lista[0]
        mini = lista[0]
        for elemento in lista:
            if elemento > maxi:
                maxi = elemento
            if elemento < mini:
                mini = elemento
    return maxi, mini
```

In [30]:
```
ll = range(20)
a, b = max_min(ll)
print a, b
```

```
19 0
```

También podemos devolver múltiples valores a través de una **lista**

Ámbito de los identificadores

In [31]:
```
from math import sqrt, asin, pi
def area_triangulo(a, b, c):
    s = (a + b + c) / 2.0 ######
    return sqrt(s * (s-a) * (s-b) * (s-c))
def angulo_alfa(a, b, c):
    s = area_triangulo(a, b, c) #######
    return 180 / pi * asin(2.0 * s / (b*c))
def menu():
    opcion = 0
    while opcion != 1 and opcion != 2:
        print '1) Calcular area del triángulo'
        print '2) Calcular ángulo opuesto al primer lado'
        opcion = int(raw_input('Escoge opción: '))
    return opcion

def main():
    lado1 = float(raw_input('Dame lado a: '))
    lado2 = float(raw_input('Dame lado b: '))
    lado3 = float(raw_input('Dame lado c: '))

    s = menu() #########
    if s == 1:
        resultado = area_triangulo(lado1, lado2, lado3)
    else:
        resultado = angulo_alfa(lado1, lado2, lado3)

    print 'Escogiste la opcion', s
    print 'El resultado es:', resultado
```

In [32]:
```
main()
```

```
Dame lado a: 3
Dame lado b: 3
Dame lado c: 3
1) Calcular area del triángulo
2) Calcular ángulo opuesto al primer lado
Escoge opción: 1
Escogiste la opcion 1
El resultado es: 3.89711431703
```

Observa que algunos identificadores se repiten (a, b, c, s), con significados distintos, pero no hay confusión porque cada uno "vive" en un entorno distinto.

CUIDADO Cada identificador debe estar definido en su entorno correspondiente. En particular, en la definición de una función sólo deben aparecer identificadores correspondientes a funciones externas, parámetros o *variables locales*. No deben aparecer variables externas.

Veamos un ejemplo incorrecto.

In [33]:
```
def opera(n):
    return n + dato #????

dato = int(raw_input('n = '))
k = int(raw_input('Dato = '))
print 'Resultado = ', opera(k)
```

```
n = 5
Dato = 6
Resultado =  11
```

La función **opera(n)** es inutilizable fuera de este contexto.

In []: