2.6 Funciones

En capítulos anteriores hemos aprendido a utilizar funciones:

- Predefinidas: *abs*, *round*, etc.
- Importadas de módulos: *sin*, *cos* del módulo *math*

Vamos ahora a aprender a definir nuestras funciones.

2.6 Funciones

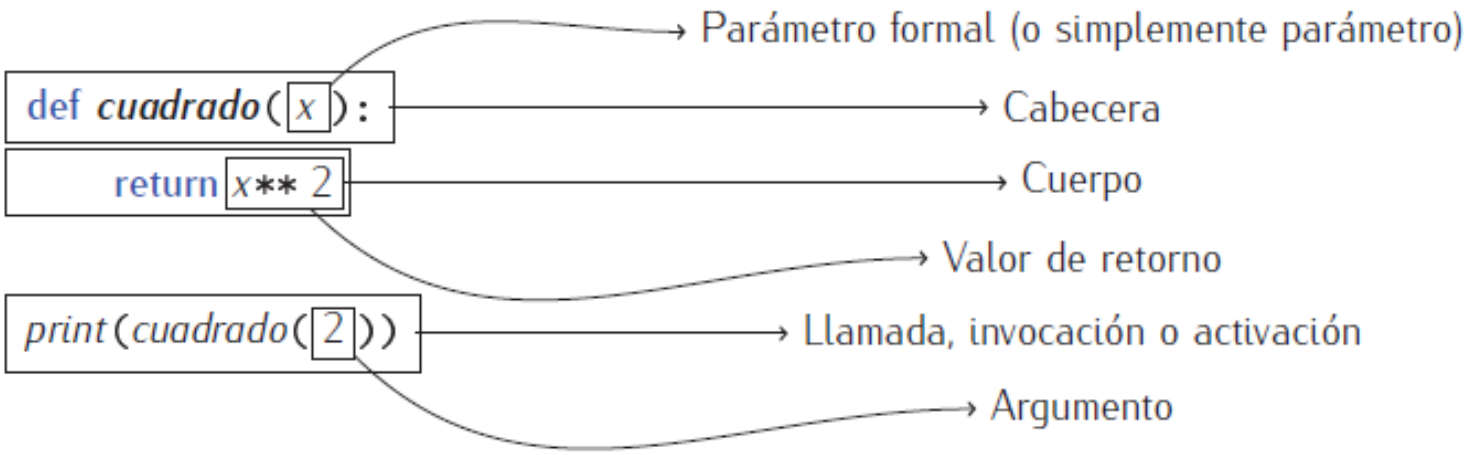
La siguiente función recibe un número y devuelve el cuadrado de dicho número:

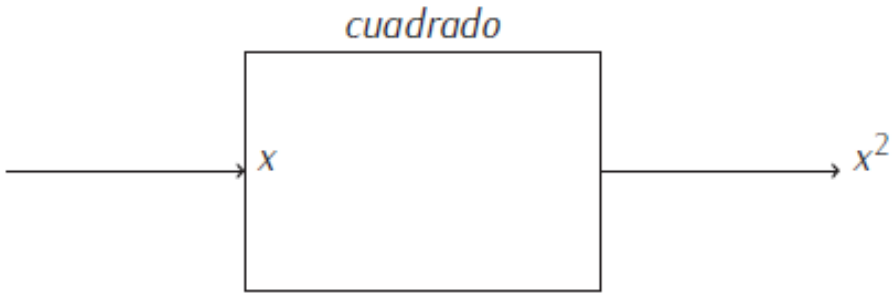
```
def cuadrado(x):
    return x ** 2

n=4
print('El cuadrado de {} es {}'.format(n,cuadrado(n))
```

Nuestros programas definirán funciones y las llamarán

2.6 Funciones





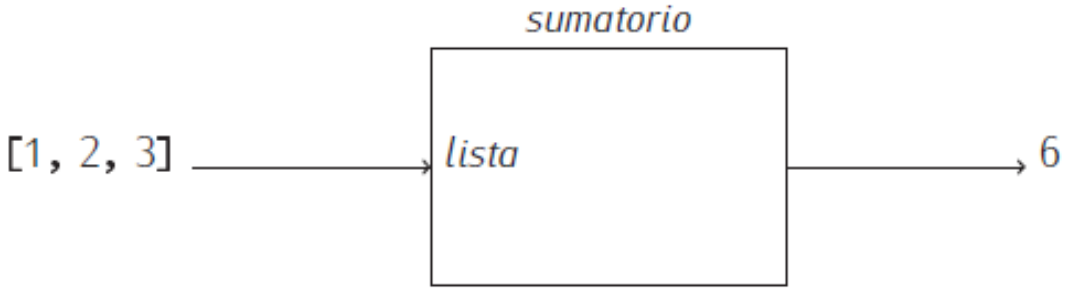
Las reglas para dar nombre a las funciones y a sus parámetros son las mismas que seguimos para dar nombre a las variables

2.6 Funciones

Otro ejemplo:

suma_lista.py

```
1 def sumatorio(lista):
2     suma = 0
3     for número in lista:
4         suma += número
5     return suma
6
7 a = [1, 2, 3]
8 print(sumatorio(a))
```



Aunque hemos aprendido a calcular sumatorios con bucles, desde la versión 2.3, Python ofrece una forma mucho más cómoda de hacerlo: la función predefinida ***sum***

2.6 Funciones

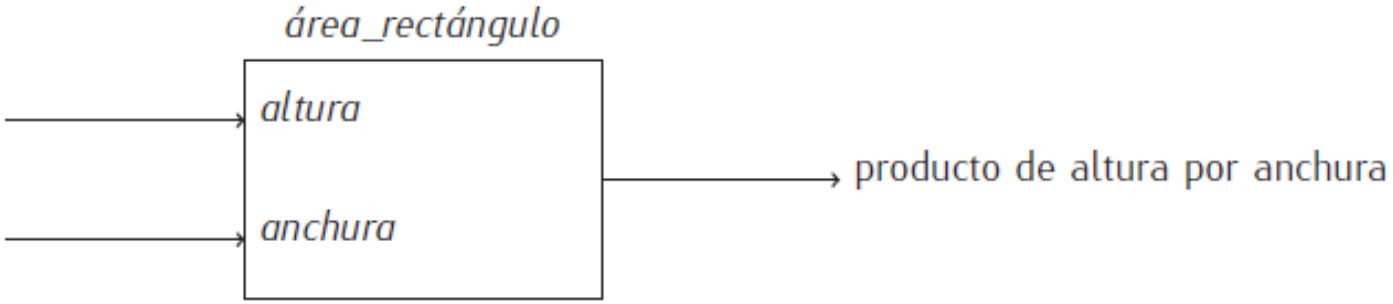
Otro ejemplo: Este programa calcula el máximo de los elementos de una lista de números.

maximo.py

```
1 def máximo(lista):
2     if len(lista) > 0:
3         candidato = lista[0]
4         for elemento in lista:
5             if elemento > candidato:
6                 candidato = elemento
7     else:
8         candidato = None
9   return candidato
```

2.6 Funciones

También se pueden definir funciones con varios parámetros



rectangulo.py

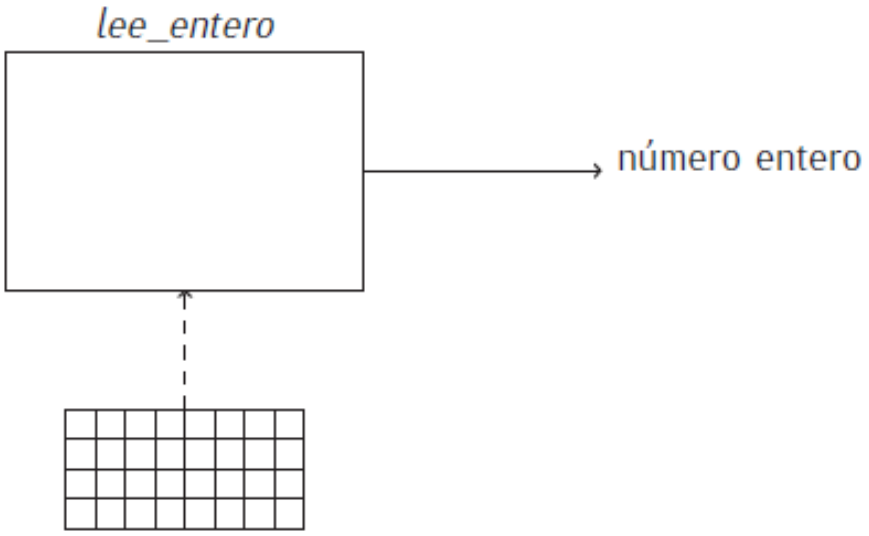
```
1 def área_rectángulo(altura, anchura):
2     return altura * anchura
3
4 print(área_rectángulo(3, 4))
```

2.6 Funciones

No es obligatorio utilizar parámetros en las funciones:

lee_entero.py

```
1 def lee_entero():
2     return int(input())
3
4 a = lee_entero()
```



2.6 Funciones

Un ejemplo típico son los menús que leen opciones:

funcion_menu.py

```
1  def menú():
2      opción = ''
3      while not (opción >= 'a' and opción <= 'c'):
4          print('Cajero␣automático.')
5          print('a)␣Ingresar␣dinero.')
6          print('b)␣Sacar␣dinero.')
7          print('c)␣Consultar␣saldo.')
8          opción = input('Escoja␣una␣opción:␣')
9          if not (opción >= 'a' and opción <= 'c'):
10             print('Solo␣puede␣escoger␣a,␣b␣o␣c.␣Inténtelo␣de␣nuevo.')
11     return opción
```

Que se llamaría de la siguiente forma:

```
1  acción = menú()
```

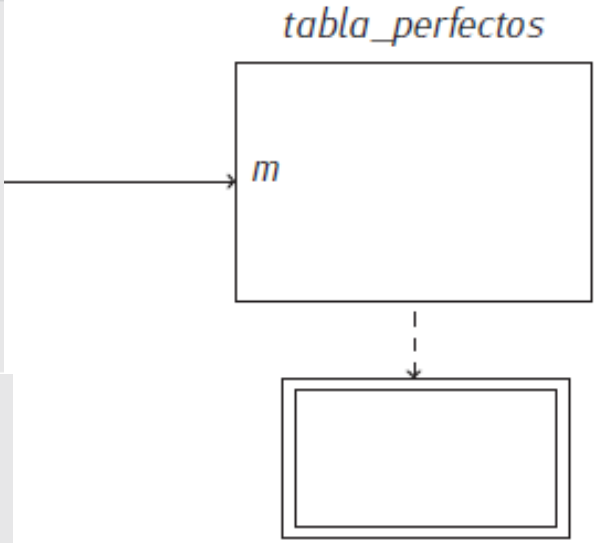
2.6 Funciones

Existen funciones que no devuelven valores, por ejemplo, las que se utilizan para escribir valores en pantalla:

Ejemplo: Solicita al usuario un número y muestra por pantalla todos los números perfectos entre 1 y dicho número.

tabla_perfectos.py

```
1  def es_perfecto(n): # Averigua si el número n es o no es perfecto.
2      sumatorio = 0
3      for i in range(1, n):
4          if n % i == 0:
5              sumatorio += i
6      return sumatorio == n
7
8  def tabla_perfectos(m): # Muestra todos los números perfectos entre 1 y m.
9      for i in range(1, m+1):
10         if es_perfecto(i):
11             print(i, 'es un número perfecto')
12
13 número = int(input('Dame un número: '))
14 tabla_perfectos(número)
```



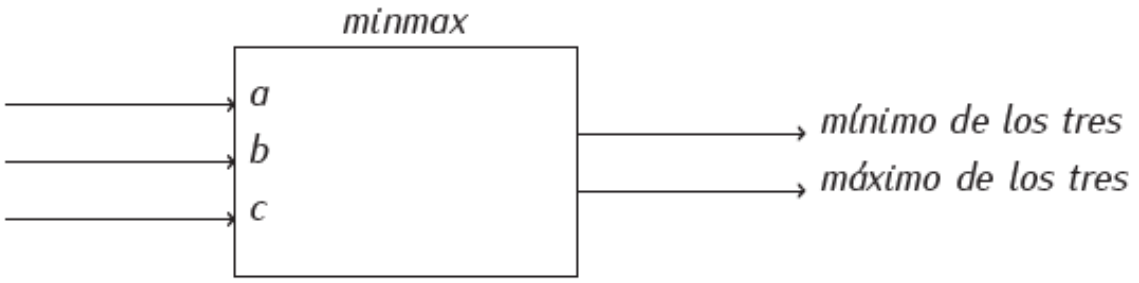
Esta funciones no tienen sentencia ***return.***

2.6 Funciones

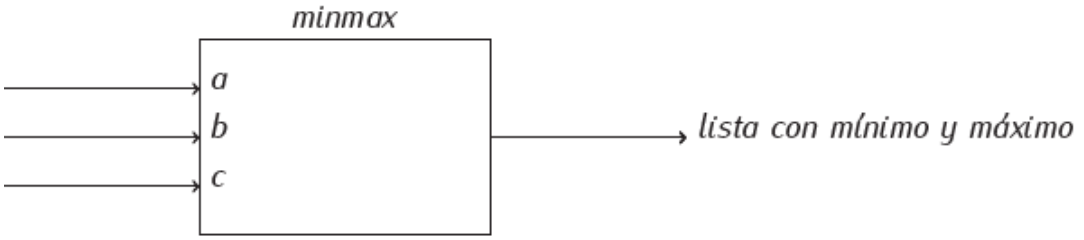
Una función puede devolver varios valores mediante una lista:

minmax.py

```
1  def minmax(a, b, c):
2      # Calcular el mínimo
3      if a < b:
4          if a < c:
5              min = a
6          else:
7              min = c
8      else:
9          if b < c:
10             min = b
11         else:
12             min = c
13
14     # Calcular el máximo
15     if a > b:
16         if a > c:
17             max = a
18         else:
19             max = c
20     else:
21         if b > c:
22             max = b
23         else:
24             max = c
25
26     return [min, max]
```



Más apropiado:



Llamada: [a, b]= minmax(10, 2, 5)

print('El mínimo es ', a)

print('El máximo es ', b)

2.6 Funciones

Variables globales y locales

Vamos a estudiar la diferencia entre las variables definidas en el programa principal y dentro de las funciones.

Ejemplo: Definamos una función que, dados los tres lados de un triángulo, devuelva el valor de su área.

$$\sqrt{s(s-a)(s-b)(s-c)} \qquad \text{donde } s = (a+b+c)/2.$$

triangulo.py

```
1 from math import sqrt
2
3 def área_triángulo(a, b, c):
4     s = (a + b + c) / 2
5     return sqrt(s * (s-a) * (s-b) * (s-c))
6
7 print(área_triángulo(1, 3, 2.5))
```

La variable s es una variable local. *a*, *b* y *c* son también variables locales.

2.6 Funciones

Variables globales y locales

triangulo.py

```
1 from math import sqrt
2
3 def área_triángulo(a, b, c):
4     s = (a + b + c) / 2
5     return sqrt(s * (s-a) * (s-b) * (s-c))
6
7 print(área_triángulo(1, 3, 2.5))
8 print(s)
```

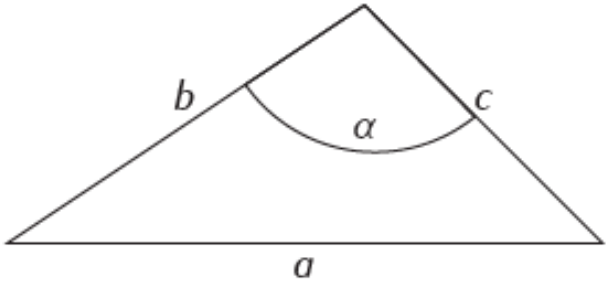
Este programa provoca un error al ejecutarse porque intenta acceder a *s* desde el programa principal.

Las variables que sólo existen en el cuerpo de una función se denominan ***variables locales***. El resto de variables se llaman ***variables globales***.

2.6 Funciones

Variables globales y locales

Otro ejemplo: Queremos ayudarnos con un programa en el cálculo del área de un triángulo de lados *a, b* y *c* y en el cálculo del ángulo (en grados) opuesto al lado *a.*



$$\alpha = \frac{180}{\pi} \cdot \arcsin\left(\frac{2s}{bc}\right)$$

donde s es el área del triángulo y arcsin es la función arco-seno.

2.6 Funciones: Variables globales y locales

area_y_angulo.py

```
1  from math import sqrt, asin, pi
2
3  def área_triángulo(a, b, c):
4      s = (a + b + c) / 2
5      return sqrt(s * (s-a) * (s-b) * (s-c))
6
7  def ángulo_alfa(a, b, c):
8      s = área_triángulo(a, b, c)
9      return 180 / pi * asin(2 * s / (b*c))
10
```

```
11 def menú():
12     opción = 0
13     while opción != 1 and opción != 2:
14         print('1) Calcular área del triángulo')
15         print('2) Calcular ángulo opuesto al primer lado')
16         opción = int(input('Escoge opción: '))
17     return opción
18
19 lado1 = float(input('Dame lado a: '))
20 lado2 = float(input('Dame lado b: '))
21 lado3 = float(input('Dame lado c: '))
22
23 s = menú()
24
25 if s == 1:
26     resultado = área_triángulo(lado1, lado2, lado3)
27 else:
28     resultado = ángulo_alfa(lado1, lado2, lado3)
29
30 print('Escogiste la opción', s)
31 print('El resultado es:', resultado)
```

-
-
- Observa que tenemos dos variables *s* diferentes y que cada una toma valores distintos sin interferir en la otra. Una es local y la otra global.

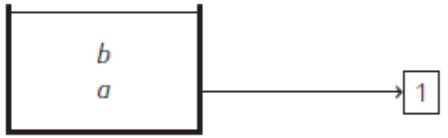
2.6 Funciones

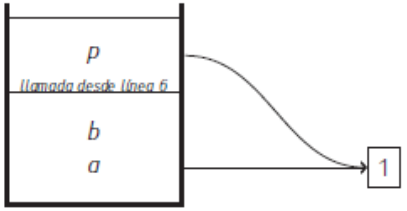
¿Qué ocurre si el parámetro es modificado dentro de la función? ¿Se modificará igualmente la variable o parámetro del ámbito desde el que se produce la llamada? Veamos un ejemplo:

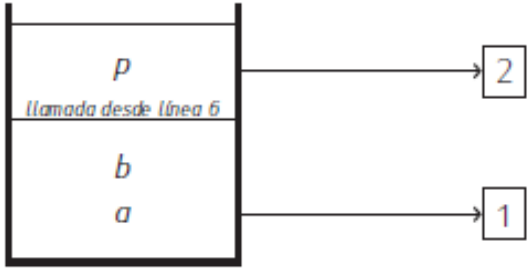
parametros.py

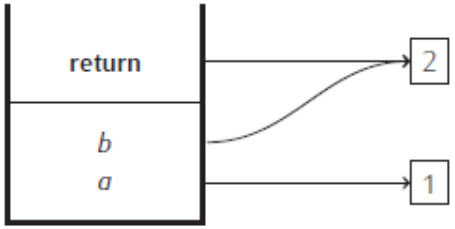
```
1 def incrementa(p):
2     p = p + 1
3     return p
4
5 a = 1
6 b = incrementa(a)
7
8 print('a:', a)
9 print('b:', b)
```

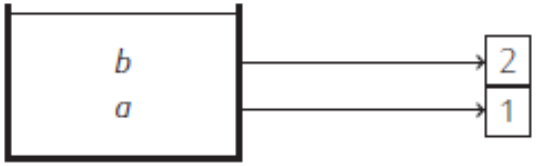
```
a: 1
b: 2
```











2.6 Funciones

¿Qué ocurre si el parámetro es modificado dentro de la función? ¿Se modificará igualmente la variable o parámetro del ámbito desde el que se produce la llamada?

parametros.py

```
1 def incrementa(a):
2     a = a + 1
3     return a
4
5 a = 1
6 b = incrementa(a)
7
8 print('a:', a)
9 print('b:', b)
```

ejercicio_parametros.py

```
1 def modifica_parámetros(x, y):
2     x = 1
3     y[0] = 1
4
5 a = 0
6 b = [0, 1, 2]
7 modifica_parámetros(a, b)
8
9 print(a)
10 print(b)
```

2.6 Funciones

- Si un parámetro modifica su valor mediante una asignación, (probablemente) obtendrá una nueva zona de memoria y perderá toda relación con el argumento del que tomó valor al efectuar el paso de parámetros.
- Números y cadenas son inmutables y, por tanto, no se modifican.
- **Cuidado:** dentro de una función se puede modificar una lista (mutable): Operaciones como *append*, *del* o la asignación a elementos indexados de listas modifican la propia lista, por lo que los cambios afectan tanto al parámetro como al argumento.

2.6 Recursión: Función factorial(x)

Factorial : x! = producto de todos los números enteros menores o iguales que x.

5! = 5 * 4 * 3 * 2 * 1

```
def factorial(x):
    F = 1
    for i in range(1,x+1):
        F *= i
    return F
```

2.6 Recursión: Función factorial(x)

Factorial : x! = producto de todos los números enteros menores o iguales que x.

5! = 5 * 4 * 3 * 2 * 1

Podemos redefinir f(n) = n! de manera recursiva?

Base: f(1) = 1

Paso recursivo: f(n) = n * f(n-1).

2.6 Recursión: Factorial function

```
def factorial(num):
    if num==1:
        return 1
    else :
        return n * factorial(n-1)

print factorial(5)
```

2.6 Recursión: Serie Fibonacci

En la serie de números de Fibonacci, cada número es la suma de los dos anteriores

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,..

Definición recursiva de los números de Fibonacci:

$f(0) = 0$

$f(1) = 1$

$f(n+2) = f(n) + f(n+1)$

2.6 Recursión: Serie Fibonacci

En la serie de números de Fibonacci, cada número es la suma de los dos anteriores

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, ...

```
def fib(n):
    fib_num = [1,1]
    for i in range(2,n):
        fib_num.append(fib_num[-1] + fib_num[-2])
    return fib_num[:n]
```

2.6 Recursión: Serie Fibonacci

```
def fibonacci(num):
        if num == 0:
                return 0
        elif num == 1:
                return 1
        else:
                return fibonacci(num - 1) + fibonacci(num - 2)

fib_numbers = [fibonacci(i) for i in range(20)]
```