Funciones

Cuando definimos una función, disponemos de una herramienta útil que podemos luego usar en múltiples situaciones concretas:

Definición y llamada de una función

Estos dos conceptos se ven mucho mejor con un ejemplo:

In [1]:

```
# Definición de una función:

def formar_numero(dec, uni):
    valor_decenas = 10 * dec
    valor_unidades = uni
    return valor_decenas + valor_unidades

# Distintas llamadas a la función anterior:

formar_numero(3, 4), formar_numero(9, 2), formar_numero(0, 5)
```

Out[1]:

```
(34, 92, 5)
```

En el ejemplo anterior, la función "componer_numero" se define usando los *parámetros abstractos* "dec" y "uni", que representan las cifras de las decenas y las unidades de un número.

El cuerpo de la definición se compone de instrucciones, que trabajan con los parámetros abstractos. Una instrucción especial, llamada *return* sirve para expresar el valor que la función devuelve, tras los cálculos.

Cada uso (o *llamada*) de la función se hace con los parámetros concretos con los que realmente necesitamos realizar los cálculos. En cada llamada, los parámetros abstractos (también llamados parámetros *formales*) asumen los valores de los parámetros concretos (también llamados parámetros *reales*) y se ejecuta el cuerpo con ellos.

Los ejemplos siguientes son fáciles de entender. En ellos, te resultará fácil distinguir entre definición y llamadas, y entre parámetros formales y reales.

In [2]:

```
def media(x, y):
    return (x + y) / 2

media(3,25), media(3.4,5.6), media(-1,-3)
```

Out[2]:

```
(14.0, 4.5, -2.0)
```

In [3]:

```
def media_4(a1, a2, a3, a4):
    s = 0.0
    s = s + a1
    s = s + a2
    s = s + a3
    s = s + a4
    return s / 4

media_4(1.5, 9.1, 3.0, 6.7)
```

Out[3]:

```
5.075
```

In [4]:

```
import math

def area_circulo(radius):
    return math.pi * radius ** 2

print(math.pi)
area = area_circulo(2)
print(area)
```

```
3.141592653589793
12.566370614359172
```

Documentación de una función

Debemos indicar lo que hace la función y el tipo que deben tener los parámetros de entrada y el valor devuelto.

In [5]:

```
def maximo(x, y):
    """
    Función que calcula el máximo de 2 números

    Parameters
    ----------
    x : int
        El primer número
    y : int
        El segundo número

    Return
    ------
    int
        El máximo de los valores x e y

    Example
    -------
    >>> maximo(2, 3)
    3
    """
    return (x + y + abs(x - y)) // 2

maximo(2, 3), maximo(2, -3), maximo(7, 7)
```

Out[5]:

```
(3, 2, 7)
```

In [6]:

```
def media_4(a1, a2, a3, a4):
    """
    Función que devuelve la media de 4 números reales

    Parameters
    ----------
    a1, a2, a3, a4: number (int or float)

    Return
    ------
    float
        la media de a1, a2, a3 y a4

    Example
    -------
    >>> media_4(1.5, 9.1, 3.0, 6.7)
    5.075
    """
    s = 0.0
    s = s + a1
    s = s + a2
    s = s + a3
    s = s + a4
    return s / 4

media_4(1.5, 9.1, 3.0, 6.7)
```

Out[6]:

```
5.075
```

Requisitos: precondición

Cuando sea necesario, también se han de poner los requisitos que deben cumplir los parámetros de una función.

In [7]:

```
import math

def lado_cuadrado(area):
    """
    Función que calcula el lado de un cuadrado, comocida su área

    Parameters
    ----------
    area : float
        El área de un cuadrado

    Returns
    ------
    float
        El lado de dicho cuadrado

    Precondition
    ------------
    area >= 0

    Example
    -------
    >>> circle(3)
    28.274333882308138
    """
    lado = math.sqrt(area)
    return lado

print(lado_cuadrado(2))

# El siguiente ejemplo fallará. Vemos que la precondición era necesaria:

print(lado_cuadrado(-2))
```

```
1.4142135623730951


---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-71833929903c> in <module>
     31 # El siguiente ejemplo fallará. Vemos que la precondición era necesa
ria:
     32
---> 33 print(lado_cuadrado(-2))

<ipython-input-7-71833929903c> in lado_cuadrado(area)
     24     28.274333882308138
     25     """
---> 26     lado = math.sqrt(area)
     27     return lado
     28

ValueError: math domain error
```

Una forma de imponer una precondición es mediante una aserción al inicio de una función:

In [8]:

```
import math

def lado_cuadrado(area):
    assert area >= 0, "el área debe ser positiva"
    lado = math.sqrt(area)
    return lado

print(lado_cuadrado(2))

# El siguiente ejemplo fallará. Vemos que la precondición era necesaria:

print(lado_cuadrado(-2))
```

```
1.4142135623730951


---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-8-490fbbca0c6a> in <module>
     10 # El siguiente ejemplo fallará. Vemos que la precondición era necesa
ria:
     11
---> 12 print(lado_cuadrado(-2))

<ipython-input-8-490fbbca0c6a> in lado_cuadrado(area)
      2
      3 def lado_cuadrado(area):
----> 4     assert area >= 0, "el área debe ser positiva"
      5     lado = math.sqrt(area)
      6     return lado

AssertionError: el área debe ser positiva
```

Como ejemplo de lo anterior, fíjate en la siguiente función. Resuelve una ecuación de segundo grado $ax^2 + bx + c = 0$, dados sus coeficientes. Lo hace áplicando la fórmula siguiente:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Pero no funciona si el radicando es negativo o si el coeficiente a es nulo.

In [9]:

```
def ec_2_grado(a, b, c):
    """
    Función que calcula las soluciones de una ecuación cuadrática
        a * x**2 + b * x + c = 0

    Parameters
    ----------
    a, b, c : float
        coeficientes de la ecuación

    Returns
    -------
    (float, float)
        Solutions of equation

    Precondition
    ------------
    a != 0 and b*b - 4*a*c >= 0

    Example
    -------
    >>> ec_2_grado(1, -5, 6)
    (3.0, 2.0)
    """
    disc = b*b - 4*a*c
    sol1 = (-b + math.sqrt(disc)) / (2*a)
    sol2 = (-b - math.sqrt(disc)) / (2*a)
    return sol1, sol2

a, b = ec_2_grado(1, -5, 6)
print(a, b)
```

```
3.0 2.0
```

¿Qué ocurrre a la función si los parámetros no cumplen la precondición?

In [10]:

```
a, b = ec_2_grado(1, 1, 4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-cc47d7b74a8a> in <module>
----> 1 a, b = ec_2_grado(1, 1, 4)

<ipython-input-9-fbc1a19a6639> in ec_2_grado(a, b, c)
     24     """
     25     disc = b*b - 4*a*c
---> 26     sol1 = (-b + math.sqrt(disc)) / (2*a)
     27     sol2 = (-b - math.sqrt(disc)) / (2*a)
     28     return sol1, sol2

ValueError: math domain error
```

In [11]:

```
a, b = ec_2_grado(0, 4, 1)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-11-4a6e2865daa0> in <module>
----> 1 a, b = ec_2_grado(0, 4, 1)

<ipython-input-9-fbc1a19a6639> in ec_2_grado(a, b, c)
     24     """
     25     disc = b*b - 4*a*c
---> 26     sol1 = (-b + math.sqrt(disc)) / (2*a)
     27     sol2 = (-b - math.sqrt(disc)) / (2*a)
     28     return sol1, sol2

ZeroDivisionError: float division by zero
```

A lo mejor puedes, tú mismo, definir una nueva versión que da lensajes más adecuados, como los siguientes:

- El discriminante de la ecuación es negativo
- La ecuación no es de primer grado

Errores comunes

Se nos olvida el `return`

In [12]:

```
def numero_triangular(n):
    """
    This function computes the n-th triangular number

    Parameters
    ----------
    n : int

    Returns
    -------
    int

    Precondition
    ------------
    n > 0
    """
    ntriag = (n * (n +1)) // 2
```

In [13]:

```
n = numero_triangular(5)
print(n)
```

```
None
```

La función devuelve el valor `None` (un valor *vacío*).

La función siguiente opera correctamente gracias a los redondeos, necesarios para limar los errores de precisión en los cálculos de las raíces.

In [14]:

```
def fibonacci(n):
    """
    This function returns the n-th Fibonacci number

    Parameters
    ----------
    n : int
        n-th Fibonacci number

    Returns
    -------
    int

    Precondition
    ------------
    n > 0

    Example
    -------
    >>> fibonacci(5)
    5
    """
    phi = (1 + math.sqrt(5)) / 2
    psi = (1 - math.sqrt(5)) / 2
    # do not forget the int(...) and round functions, otherwise it will be a real number.
    return int(round( (phi**n - psi**n) / math.sqrt(5) ))

fibonacci(1), fibonacci(2), fibonacci(3), fibonacci(4), fibonacci(5), fibonacci(6)
```

Out[14]:

```
(1, 1, 2, 3, 5, 8)
```

Referencias

Siguiendo con las referencias de w3schools, damos seguidamente el enlace sobre las funciones:

https://www.w3schools.com/python/python_functions.asp
(https://www.w3schools.com/python/python_functions.asp)