Funciones

Las funciones nos permiten agrupar el código en bloques reutilizables. De este modo, evitamos repetir innecesariamente el código, y además, podemos reutilizarlo en diferentes partes del programa.

1. Definición de funciones

A la hora de definir una función en Python, comenzamos con la palabra `def` seguida del nombre de la función y los parámetros que tendrá, entre paréntesis. Para cada parámetro sólo debemos especificar su nombre (recuerda que en Python no se especifican los tipos de datos explícitamente).

Igual que ocurre con otras estructuras como *if* o *while*, el código perteneciente a una función debe estar tabulado. Además, si queremos que la función devuelva algún valor, podemos emplear la cláusula `return` como en otros lenguajes, aunque no es obligatoria si no queremos devolver nada. También podemos definir un *return* vacío para indicar que no se devuelve nada.

Veamos algunos ejemplos.

- Esta función toma dos parámetros y devuelve el mayor de ellos

```
def maximo(num1, num2):
    if num1 > num2:
        return num1
    else:
        return num2
```

- Esta función toma un texto como parámetro y lo saca por pantalla:

```
def imprimeTexto(texto):
    print(texto)
    return # Esta línea se podría omitir
```

A la hora de llamar a estas funciones desde otras partes del código, se hace igual que en muchos otros lenguajes:

```
print ("Escribe dos números")
n1 = int(input())
n2 = int(input())
print ("El máximo es", maximo(n1, n2))

texto = input("Escribe un texto:")
imprimeTexto(texto)
```

2. Sobre los parámetros

Veamos a continuación algunos aspectos relevantes sobre los parámetros que pasamos a las funciones.

2.1. Paso por valor y por referencia

En Python todos los parámetros simples (números, booleanos y textos) se pasan por valor, con lo que no podemos modificar el valor original del dato (se pasa una copia del mismo), y todos los tipos complejos (arrays, u objetos) se pasan por referencia. Esto último implica que, siempre que se mantenga la referencia original, podemos modificar el valor del parámetro de forma persistente (se aplica a la variable original utilizada como parámetro). Por ejemplo, si empleamos esta función:

```
def anyadirValores(lista):
    lista.append(30)
    print ("Valores en la función:", lista)
    return
```

y llamamos a la función de este modo:

```
lista1 = [10, 20]
anyadirValores(lista1)
print ("Valores fuera de la función:", lista1)
```

Entonces la variable *lista1* y el parámetro *lista* almacenan los mismos valores finales: `[10, 20, 30]`. Sin embargo, si usamos esta otra función:

```
def anyadirValores(lista):
    lista = [30, 40]
    print ("Valores en la función:", lista)
    return
```

y llamamos a la función del mismo modo que antes, entonces la variable original *lista1* tendrá los valores `[10, 20]` al finalizar, y el parámetro *lista* tendrá los valores `[30, 40]` dentro de la función, pero este cambio se pierde fuera de la misma, porque se ha modificado la referencia de la variable (la hemos reasignado entera en la función), y por tanto hemos creado una nueva referencia distinta a la original, que no modifica entonces su contenido.

2.2. Tipos de parámetros

Los parámetros definidos en una función pueden ser:

- **Obligatorios**: es el modo normal. Si simplemente definimos el nombre de cada parámetro, entonces ese parámetro es obligatorio, y debemos darles valor al llamarles, en el mismo orden en que están definidos. Aquí podemos ver un ejemplo (el mismo visto anteriormente):

```
def maximo(num1, num2):
    if num1 > num2:
        return num1
    else:
        return num2
```

- **Palabras clave**: podemos utilizar el nombre original de los parámetros cuando llamamos a la función, y de este modo no tenemos por qué seguir el mismo orden que cuando se definió dicha función. Por ejemplo:

```
def imprimirDatos(nombre, edad):
    print ("Tu nombre es", nombre, "y tu edad es", edad)
    return
...
imprimirDatos(edad = 28, nombre = "Juan")
```

- **Valores por defecto**: cuando declaramos la función con sus parámetros, podemos asignar un valor por defecto a los parámetros que queramos. Así, si queremos llamar a la función, podemos omitir los parámetros que tengan un valor por defecto asignado, si queremos. Por ejemplo:

```
def imprimirDatos(nombre, edad = 0):
    print ("Tu nombre es", nombre, "y tu edad es", edad)
    return
    ...
imprimirDatos("Juan") # Imprime "Tu nombre es Juan y tu edad es 0"
```

> **NOTA**: es importante que los parámetros que tengan valores por defecto se coloquen todos al final de la lista de parámetros (tras los obligatorios), para que así no queden huecos si queremos llamar a la

> función omitiendo parámetros. También es importante que, cuando omitamos un parámetro, los que vayan detrás también se omitan para que no se desplace el orden y se asignen por error a otros parámetros.

- **Longitud variable**: podemos especificar como último parámetro de la función un tipo especial que permite pasar tantos parámetros como necesitemos. Por ejemplo:

```
def imprimirTodo(num1, *numeros):
    print("Primer número:", num1)
    for num in numeros:
        print num
    return
```

> **Ejercicio 1**:
>
> Crea un programa llamado `Funciones.py` con las siguientes funciones:
>
> 1. Una función llamada `fibonacci` que reciba un entero *n* como parámetro y devuelva el n-ésimo elemento de la serie de Fibonacci
> 2. Una función llamada `esPrimo` que reciba un número como parámetro y devuelva un booleano indicando si el número es primo o no
>
> Desde el programa principal, llama a la función `fibonacci` para mostrar el 5º y 10º números de la serie de Fibonacci, y usa la función `esPrimo` para sacar los números primos que haya del 1 al 50.

2.3. Paso de parámetros al programa principal

A pesar de que en Python no existe una función principal `main` como la que sí existe en otros lenguajes como C, Java, C#... sí es posible pasar parámetros al programa desde el terminal cuando lo ejecutamos. Para ello, importamos el elemento `sys`, que hace referencia al sistema sobre el que se ejecuta el programa. Dentro, disponemos de un array predefinido llamado `argv`, similar al que existe en C o C++, con los datos que le llegan al programa. El primero de ellos, igual que ocurre en C o C++ es el nombre del propio ejecutable, y el resto son los parámetros adicionales.

```
import sys

for i in range(1, len(sys.argv)):
    # Recorremos los parámetros quitando el 0 (ejecutable)
    print(sys.argv[i])
```