Funciones en Python

Introducción a Python

José Miguel Gimeno

Funciones en Python

Recordemos el concepto de función:

- Fragmento de código con un nombre asociado
- Recibe cero o más **argumentos** como entrada
- Realiza una secuencia de instrucciones
- Devuelve un valor OR realiza una tarea (concepto de función versus procedimiento)

Funciones en Python

Qué ventajas tiene usar funciones:

- Reutilización de código
- Abstracción
- Estructuración

Definición de funciones en Python

- Pueden crearse en cualquier punto del programa
- Es obligatorio definirlas antes de llamarlas

Definición de funciones en Python

La sintaxis consta de:

- Palabra reservada **def**
- Nombre de la función (recomendación)
- Paréntesis, que pueden incluir o no parámetros
- Dos puntos, que indican el final de la definición

Definición de funciones en Python

```
def nombre(parámetros):
    '''docstring'''
    resto de instrucciones
    return expresion
```

Definición de funciones en Python

- El **docstring** es importante porque sirve como documentación, es un comentario que define qué hace la función
- Las instrucciones que forman parte de la función deben tener una sangría de 4 espacios
- El **return** no es obligatorio pero puede ser muy útil

Definición de funciones en Python

Además, de acuerdo con PEP8 antes de la definición tendría que haber dos líneas en blanco

Solo es una guía de estilo, no es obligatorio, pero seguiremos esta recomendación en nuestro código.

Llamada de funciones en Python

Si intento llamar a una función antes de ser creada, se devuelve un error **NameError** advirtiendo que la función en cuestión **is not defined**

Docstring

- Se sitúa inmediatamente debajo de la definición
- Sangría 4 espacios como el resto del bloque
- Puede constar de una o varias líneas
- Se comienza y termina con triples comillas

Docstring

Para acceder al contenido del docstring cada función definida contiene un atributo denominado __doc__ donde se almacena el comentario del docstring.

Funciones, clases y módulos deben documentarse convenientemente

Docstring

- Acceder a través del atributo __doc__
- Acceder a través de help() en el modo interactivo
- Acceder a través de pydoc3 sin entrar en el intérprete (para salir utilizar q)

Parámetros

- Si los tiene, se escriben entre paréntesis y separados por comas
- Se utilizan como variables locales (no pueden usarse fuera de la función)

Órden de los parámetros

- Por defecto, se sigue el orden de la definición, pero puede cambiarse
- Para cambiarlo podemos usar el método clave=valor para establecer el orden que queramos

Parámetros por defecto

- Podemos especificar unos valores predefinidos para usarse cuando no se especifique algún parámetro
- Se indica en la definición: parámetro=valor por defecto
- Si solo se indican valores por defecto para algunos parámetros, estos deben ir al final, si no, obtenemos un SyntaxError

Parámetros múltiples indefinidos

- Cuando una función espera recibir un número de parámetros indefinido, se reciben como una tupla
- Para definir parámetros arbitrarios se antepone un asterisco al parámetro

Parámetros múltiples por clave-valor

- Para recibir un número indeterminado de parámetros por nombre se crea un diccionario dinámico anteponiendo dos asteriscos al parámetro

Ámbitos

Normalmente una función solo se puede comunicar fuera de la misma con los parámetros de entrada y con return

Los espacios de nombres de fuera y dentro de la función se conocen como ámbitos

Ámbitos

Los ámbitos son necesarios para evitar que objetos definidos en una función con el mismo nombre sobrescriban el espacio de nombres global

Ámbito local

Cada función genera su propio espacio de nombres cuando es llamada, cada uno de ellos es un ámbito local

Con la función **locals()** obtenemos los nombres de este ámbito en forma de diccionario

Ámbito local

Con la función **dir()** sin argumentos obtenemos los nombres de este ámbito en forma de lista

Ámbito global

Es el espacio de nombres del intérprete de Python

Con la función **globals()** obtenemos los nombres de este ámbito en forma de diccionario

Ámbito global

Con la función **dir()** sin argumentos obtenemos los nombres de este ámbito en forma de lista

Ámbito no local

Si el programa solo contiene funciones sin otras funciones anidadas, todas las variables libres son globales

En caso contrario, las variables libres de las subfunciones pueden ser globales o no locales

Búsqueda de nombres en los ámbitos

Si a una variable no se le asigna valor dentro de una función, Python la considera libre y busca su valor en los niveles superiores de abajo arriba

- Si a la variable se le asigna un valor en algún nivel intermedio, la variable se considera no local
- Si a la variable se le asigna un valor en el programa principal se le considera global

global

Cuando queremos que una variable creada dentro de una función tenga carácter global se usa la palabra **global**

Hay que llamar primero a la función y después a la variable, si no, nos dará un error

global

Cuando queremos que una variable creada dentro de una función tenga carácter global se usa la palabra **global**

Hay que llamar primero a la función y después a la variable, si no, nos dará un error

return

Cuando queremos acceder al resultado de una función usamos la palabra reservada return

Retorno múltiple

Cuando queremos que la función devuelva más de un valor utilizamos return y separamos por comas cada variable o expresión que queramos devolver

Ejercicio propuesto 1

Crea un programa en Python que muestre los divisores para cada número entre 1 y 47.

P.ej. para 12

12 : (1, 2 , 3, 4, 6, 12)

Debes crear y utilizar una función divisores(n) que devuelva una tupla con los divisores de n.

Ejercicio propuesto 2

Crea un programa en Python que muestre los números primos hasta el 100.

Un número primo es aquel que solo es divisible por sí mismo y por 1. El 1 no se considera primo.

Debes crear y utilizar una función primos(n) que devuelva una tupla con los números primos hasta el n.

Ejercicio propuesto 3

Crea un programa en Python que determine los números perfectos que hay entre 1 y 1000.

Un número es perfecto si la suma de sus divisores menores que el número es igual al propio número. P.ej. 6 es perfecto porque 6 = 1 + 2 + 3

Debes crear y utilizar una función perfecto(n) que devuelva n si es perfecto.

Funciones de orden superior

Pueden recibir como parámetros otras funciones

Pueden devolver funciones como resultado

filter y map

Se trata de funciones de orden superior incorporadas en el propio intérprete de Python

Son muy útiles para trabajar con listas

filter() filtra un iterable elimando los elementos que no cumplen con la condición

map() recibe una función y un iterable y devuelve un nuevo iterable con la función aplicada a cada argumento

Funciones lambda

Es posible definir funciones en una sola línea mediante el uso de expresiones lambda con la sintaxis:

Variable = lambda parámetros instrucción

A estas funciones se les llama funciones lambda o anónimas y se utilizan para agilizar el desarrollo o por claridad

Ejercicio propuesto 4

- Dada la lista numeros = [[57, 12, 94, 44, 19],[58, 84, 15, 76, 44],[48, 58, 92, 81, 63],[44, 57, 19, 94, 12]]

Crea una función convencional que calcule la media de una lista de números y usa map() para hallar la media de cada lista.

Luego reescribe el código reemplazando la función por una expresión lambda dentro de la llamada a map()

Funciones recursivas

- Se trata de funciones que se llaman a sí mismas en tiempo de ejecución
- Es útil para resolver problemas sustituyendo a bucles
- Ejemplos típicos: sumatorio y factorial

Ejercicio propuesto 5

Se trata de adivinar un número del 1 al 10. Utiliza una función recursiva

- Si se falla, la salida mostrará si el número introducido es mayor o menor que el que hay que adivinar.
- Si se acierta, la salida mostrará un mensaje de enhorabuena e indicará en qué intento se ha acertado.

yield – Funciones generadoras o iteradores

- Cuando queremos crear una lista a través de una función, primero creamos una lista vacía y luego la llenamos con un bucle
- La diferencia fundamental entre una función normal y una generadora es que no hace falta disponer de todos los elementos de la lista, solamente hace falta saber cómo generar el siguiente elemento. Así, no necesita reservar espacio con antelación.

yield – Funciones generadoras o iteradores

- Ejemplo: función normal que genera números pares
- Para construir una función generadora usamos yield en vez de return. Esto hace que se devuelva un valor, pero, además, congela la ejecución de la función hasta que se le pida otro valor.

yield – Funciones generadoras o iteradores

- No solo son útiles para generar listas, de hecho, el resultado no es una lista, sino una secuencia iterable.
- La función iter() permite convertir cadenas y otras colecciones a iteradores y trabajar de la misma forma.

Ejercicio propuesto 6

Implementa un generador de números de la secuencia Fibonacci utilizando yield.

Recuerda que los dos primeros elementos son 0 y 1 por definición y el resto se calculan como suma de los dos últimos elementos.

0, 1, 1, 2, 3, 5, 8, …