Python Programming

Módulo 1

Funciones propias

Funciones en Python

Anteriormente hemos usado funciones nativas que vienen con Python, como `len()` para calcular la longitud de una lista, o `type()` para saber el tipo de dato. Pero al igual que en otros lenguajes de programación, también podemos definir nuestras propias funciones. Para ello hacemos uso de la palabra reservada `def`.

```
def funcion():
    pass
```

La función anterior no hace nada. La palabra reservada `pass` sirve para rellenar un bloque de código requerido sintácticamente (no podría estar vacío porque lanzaría un error de sintaxis).

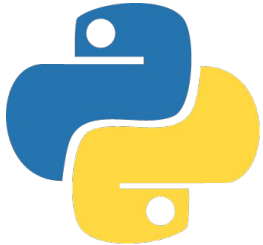
La palabra reservada `pass` no solo se emplea para completar código de bloques vacíos en funciones, también se la emplea para completar bloques vacíos de sentencias de control. Y para completar clases vacías en orientación a objetos.

Anatomía de una función

```
def nombre_funcion(argumentos):
    # código
    return retorno
```

Las funciones que definimos permiten agrupar un bloque de código o conjunto de instrucciones. Opcionalmente, reciben uno o más argumentos y retornan un valor.

En Python, para crear una función empleamos la palabra reservada `def` seguido de su nombre y sus parámetros entre paréntesis, en caso de tener. A continuación, se coloca su contenido utilizando una indentación.



Vamos a crear un ejemplo de nombre sumar con dos parámetros cuyos nombres son a y b, y que retorne el resultado de la suma de ambos.

```
def sumar(a, b):
     c = a + b
     return c
```

El uso del `return` permite salir de la función (terminarla) y transferir la ejecución de vuelta a donde se realizó la llamada. Se puede devolver uno o varios parámetros, pero una vez que se llama al `return` se detiene la ejecución de la función y se vuelve o retorna al punto donde fue llamada. Cuando una función no retorna ningún valor (es decir, no incluye la palabra reservada `return`), por defecto retorna `None`.

Ejemplo:

```
def sumar(a, b):
     print( a + b)

print(sumar(10,20))

out: 30
out: None
```

Los nombres de las funciones en Python siguen las mismas reglas que los nombres de las variables, se escriben en minúscula y con guiones bajos en lugar de espacios vacíos.

Argumentos por defecto

Para indicar parámetros opcionales, indicamos su valor por defecto con el signo de igual (**=**).

```
def imprimir_mensaje(mensaje="No has escrito nada."):
      print(mensaje)

imprimir_mensaje("Hola mundo")
imprimir_mensaje()

out: Hola mundo
out: No has escrito nada.
```

Los argumentos con valores por defecto siempre deben ir **al final de la función.** Por ejemplo, el siguiente código es inválido:

```
def sumar(a=5, b):
      return a + b
```

A los argumentos con valores por defecto se los conoce como *keyword arguments* o **"argumentos por nombre"**, ya que al llamar a la función se puede especificar su valor indicando su nombre, como se ve en el ejemplo a continuación.

```
def sumar(a=5, b=10):
      return a + b

sumar()
out: 15

sumar(b=50)
out: 55

sumar(7, 5)
out: 12

sumar(a=7, b=5)
out: 12
```

Esto permite que en la segunda llamada **a** mantenga su valor por defecto, mientras que **b** obtenga el número 50.

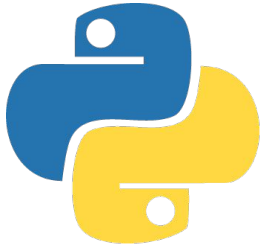
En el caso que se especifiquen ambos argumentos, indicar o no su nombre es indistinto.

Documentación en funciones

Ahora que ya tenemos nuestras propias funciones creadas, muchas veces en algún proyecto debamos compartirlas con otros programadores, para que sean parte de un programa complejo.

Leer código ajeno no es tarea fácil, por eso es importante documentar las funciones.

```
def sumar(a, b):
    """
    Descripción de la función. Como debe ser usada,
    que parámetros acepta y que devuelve
    """
    return a+b
```



Para crear un texto que nos diga qué hace la función podemos crear un comentario del tipo párrafo usando la triple comilla **"""** al principio de la función. Es únicamente al principio no en otro lado. Esto no forma parte del código, es un simple comentario un tanto especial, conocido como *docstring*.

Ahora cualquier persona que haga uso de nuestra función, podrá acceder a la ayuda de la función con `help()` y obtener las recomendaciones nuestras de cómo debe ser usada.

```
help(sumar)
```

¡Sigamos trabajando!