Funciones

Definición y características

Una función es un **bloque de código** con un **nombre** asociado, que recibe cero o más **argumentos** como entrada, sigue una **secuencia de sentencias**, la cuales ejecuta una **operación** deseada y **devuelve** un valor y/o **realiza una tarea**, este bloque **puede ser llamados** cuando se necesite.

El uso de funciones es un componente muy importante del paradigma de la programación **modular**, y tiene varias ventajas:

- **modularización**: permite segmentar un programa complejo en una serie de partes o módulos más simples, facilitando así la programación y el depurado.
- **reutilización**: permite reutilizar una misma función en distintos programas.

Python dispone de una serie de funciones **integradas** al lenguaje, y también permite crear funciones definidas por el usuario para ser usadas en su propios programas.

Funciones integradas

En python las funciones se organizan en **bibliotecas** o **módulos** que podemos incluir en nuestro programas. En Python, los módulos son simplemente archivos con la extensión «.py» que contienen código Python que se pueden importar dentro de otro programa Python.

Además de las funciones que podemos importar, el interprete Python tiene un número de funciones integradas (built-in) dentro del **módulo** `__builtins__`, las cuales están siempre disponibles. Algunas de ellas ya las hemos utilizado: `print()`, `input()`, `id()`, `range()`, `round()`, `abs()`, `type()`. Podemos ver todas las funciones integradas en Python en el siguiente enlace

También hemos utilizado funciones que hemos importado de módulos, como por ejemplo `randint()` del módulo 'random'

El objetivo de estos apuntes es que aprendamos a crear y utilizar nuestras propias funciones.

La sentencia def

Para poder utilizar una función, previamente debe estar definida. Para definir una función usamos la sentencia `def`. Su sintaxis es la siguiente:

```
def nombre(lista_de_parámetros):
    """docstring_de_función"""
    sentencias
    return [expresion]
```

La primera línea contiene la defincón de la función

- `nombre`, es el nombre de la función.
- `lista_de_parametros`, es la lista de parámetros que puede recibir una función. Puede ser vacia.

A continuación, indentado, se incluye el bloque de código de la función que puede tener:

- `docstring_de_función`, es la cadena de caracteres usada para documentar la función. Es el texto que se devuelve si pido ayuda de una función. Su inclusión es opcional.
- `sentencias`, es el bloque de sentencias en código fuente Python que realizar cierta operación dada.
- `return`, es la sentencia que permite devolver valores por parte de pla función o simplemente terminar la ejecución de la misma. Es opcional.
- `expresion`, es la expresión o variable que devuelve la sentencia `return`.

Mi primera función

Un ejemplo simple de función esta seguidamente:

```
def hola(arg):
    """El docstring de la función"""
    print "Hola", arg, "!"
...
hola("Mundo")
```

El resultado de ejecutar este programa es:

```
Hola Mundo !
```

La palabra reservada `def` se usa para definir funciones. Debe seguirle el nombre de la función en el ejemplo anterior `hola()` y la lista de parámetros formales entre paréntesis. Las sentencias que forman el cuerpo de la función empiezan en la línea siguiente, y deben estar indentado.

La primer sentencia del cuerpo de la función puede ser opcionalmente una cadenas de caracteres literal; esta es la cadenas de caracteres de documentación de la función, o `docstrings`.

Hay herramientas que usan las `docstrings` para producir automáticamente documentación en línea o imprimible, o para permitirle al usuario que navegue el código en forma interactiva; es una buena práctica incluir `docstrings` en el código que uno escribe, por lo que se debe hacer un hábito de esto.

La ejecución de la función `hola()` muestra la impresión de un mensaje **Hola Mundo !** que se imprimeen el terminal.

Argumentos y parámetros

Orden de los argumentos

Al **definir** una función los valores los cuales se reciben se denominan **parámetros**, pero durante la llamada los valores que se envían se denominan **argumentos**.

Cuando enviá argumentos a una función, estos se reciben por el **orden** en que están definidos:

```
def resta(a, b):
    print(a - b)
resta(30, 10)
resta(10, 30)
```

El resultado sería:

```
20
-20
```

Llamada sin argumentos

Si llamamos a una función que tiene parámetros definidos, sin pasarle alguno de los argumentos obtenemos error

```
def resta(a, b):
    print(a - b)
resta()
```

El resultado sería:

```
resta()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: resta() takes exactly 2 arguments (0 given)
```

Parámetros por defecto

Podemos definir el valor por defecto de los parámetros para el caso de que no le pasemos alguno de los argumentos a la función:

```
def resta(a = 0, b = 0):
    print(a - b)
resta()
resta(4)
```

En este caso no obtendríamos error, sino que el resultado sería:

```
0
4
```

Número de parámetros indeterminados

En ocasiones nos puede interesar definir funciones que permitán un número flexible de parámetros. Por ejemplo, la siguiente función nos calcula el valor medio de dos valores:

```
def media(x, y):
    resultado = (x + y) / 2
    print(resultado)
media(4, 6)
```

Obtenemos:

```
5
```

Pero que pasa si no sabemos a priori el número de valores para los que vamos a calcular la media. Sería más interesante que la función calculara la media de cualquier cantidad de valores. Para ello podemos usar como parámetro `*args`

```
def listado(*args):
    print(args)
listado(1, 2, 3, 4)
```

El resultado sería:

```
(1 , 2, 3, 4)
```

`*args` almacena la dirección de los argumentos que pasamos como parámetros. Python agrupa dichos parámetros separados por comas en un objeto, cuyo tipo de datos es una **t-upla**.

Podemos recorrer dentro de la función el listado de parámetros de la misma forma que recorremos una lista o un range. Usando `for ... in ...`

```
def listado(*args):
    for i in args:
        print(i)
listado(1, 2, 3, 4)
```

la salida sería:

```
1
2
3
4
```

Con lo que acabamos de ver, ya podríamos hacer una función que nos calcule el valor medio para un número indeterminado de parámetros:

```
def calcula_media(*args):
    n_valores = suma = 0
    for i in args:
        suma += i
        n_valores += 1
    print(suma / n_valores)
media(1, 2, 3, 4, 5)
```

la salida sería:

```
3
```

La función len()

La función integrada en el interprete `len()` en general nos devuelve el número de elementos de un objeto. Si le pasamos como parámetro una cadena nos devuleve el número de caracteres que contiene, si le pasamos un rango o una t-upla nos devuelve el número de elementos que contiene.

```
>>> print(len("aeiou"))
5
>>> print(len((1, 2, 3, 4, 5, 6)))
6
>>> print(len(range(10)))
10
```

En la función `calcula_media()`, del ejemplo anterior usamos la variable de tipo **contador** `n_valores` para almacenar el número de parámetros que se le pasan a la función. Podemos hacer lo mismo utilizando la función `len()`

```
def calcula_media(*args):
    suma = 0
    for i in args:
        suma += i
    print(suma / len(args))
```

Sentencia return

Mediante `return` las funciones pueden comunicarse con el exterior y devolver valores. Si una función no incluye `return` termina y devuelve el control al bloque de código que la llamó:

```
def saluda(nombre):
    print(f"Hola {nombre}")
print("Inicio")
saluda("amigo")
print("¡Adios!")
```

El resultado sería:

```
Inicio
Hola amigo
¡Adios!
```

La sentencia `return` en caso de aparecer no tiene por qué estar al final de la función. O puede aparecer varias veces.

```
def saluda(nombre):
    if nombre == "":
        print("Nadie a quién saludar")
        return
    print(f"Hola {nombre}")
    return

saluda("")
saluda("amigo")
```

El resultado sería:

```
Nadie a quién saludar
Hola amigo
```

Usando return para devolver valores

En la función anterior no se devuelve nada y la acción (mostrar el saludo) se realiza dentro de la función.

La sentencia `return` puede ir acompañada de parámetros que nos permiten devolver una salida a la parte del programa en la que se llama a la función.

Para el ejemplo anterior de la función `saluda()`, de forma alternativa podríamos devolver el texto a mostrar y mostrarlo fuera de la función:

```
def saluda(nombre):
    if nombre == "":
        return "Nadie a quién saludar"
    return f"Hola {nombre}"

print(saluda(""))
print(saluda("amigo"))
```

El resultado sería:

```
Nadie a quién saludar
Hola amigo
```

> **Nota**: fíjate que el valor retornado por la función queda en la posición en la que la función es invocada.

Lo mismo podríamos hacer con la función que calcula el valor medio de una lista de valores:

```
def calcula_media(*args):
    suma = 0
    for i in args:
        suma += i
    resultado = suma / len(args)
    return resultado
media = calcula_media(1, 2, 3)
print(media)
```

Obtendríamos:

```
2
```

Referencias

- Apuntes funciones en Python - Jorge de los Santos
- Funciones - unipython.com
- Funciones - Entrenamiento básico con Python
- Introducción a la programación con Python. Funciones - Mclibre

tags: `pro` `ut1` `python` `funciones`