


Open · Opened 4 days ago by  **Rubén Montero**

# Funciones que funcionan

## Resumen

- Veremos las diferencias entre Java y Python a la hora de definir funciones
- Hablaremos de la *indentación* en Python
- Añadiremos un nuevo fichero de código fuente `.py` en `intro/`
- Veremos qué es `pass`
- Definiremos tres funciones

## Descripción

Ya somos familiares con los conceptos de [procedimiento y función](#). Recordemos que en Java se *declaran*, por ejemplo, así:

```
public void doSomething() { /* ... */ }

public String getMyMessage(int someValue, float otherValue) { /* ... */ }
```

Veremos cuántas diferencias existen en Python a la hora de declarar procedimientos ó funciones:

### 1) No hace falta que estén en una clase

Aún no hemos visto *clases* en Python.

No necesitamos hacerlo para entender *cómo* se usan las funciones ó procedimientos. (Por brevedad, diremos sólo '*funciones*' a partir de ahora)

¡No hace falta que exista ninguna clase para *declarar* e *invocar* funciones!

### ¿Pero puede haber funciones dentro de una clase?

Sí, claro. En Python las clases podrán tener métodos. Hay alguna peculiaridad que veremos más adelante.

### 2) No hay que especificar visibilidad

Ya nos podemos olvidar de `public` y `private`.

En Python, todo es público.

### 3) Se usa la palabra `def`

**Siempre.** Es una palabra reservada del lenguaje:

```
def ...
```

### 4) No hay que especificar el *tipo* de dato devuelto

Mientras que en Java solíamos indicar `void`, `int`, `ArrayList<String>`, ... En **Python no hay que hacerlo**.

### 5) Igual que en Java, se escribe el *nombre* de la función y paréntesis

Por ejemplo:

```
def my_function()

def other_function()
```

(Nótese que usamos `snake_case` en vez de `LowerCamelCase`)

### ¿Y los parámetros?

Bien visto.

En Java, los declarábamos separados por comas ( , ), indicando su **tipo** como prefijo:

```
public void myFunction(String param1, int param2)
```

En Python, los declaramos separados por comas ( , ) si hay más de uno, pero **no** indicamos su tipo:

```
def my_function(param1, param2)
```

Si un parámetro no se corresponde con lo que esperamos... ¡Ya se verá en tiempo de ejecución!

## 6) Se usan dos puntos ( : ), no llaves ( { } )

Basta con indicar con dos puntos, al final, que la declaración de la función ha terminado:

```
def my_function(param1, param2):
```

¡Así de fácil!

Un momento...

## Entonces, ¿cómo sé dónde termina la función?

Buena pregunta.

Si no hay una llave ( } ) que *cierre* la función... ¿Qué líneas pertenecen a la siguiente función y cuáles no?

```
def my_function(param1, param2):  
    print(param1)  
    print(param2)  
    print("Hello")
```

**No** se puede distinguir. De hecho, este código es *incorrecto*.

La respuesta...

...no es con una línea en blanco u otro carácter de cierre.

Es:

## Indentación

La [indentación](#) ó sangría consiste en *tabular* (añadir espacio) a ciertas líneas para aumentar la legibilidad.

En Java, estamos acostumbrados a *indentar* el contenido de una función, clase, bucle...

```
public void dowhatever() {  
    System.out.println("whatever");  
}
```

En Python, estamos **obligados**:

```
def my_function(param1, param2):  
    print(param1)  
    print(param2)  
    print("Hello")
```

Así, sabemos dónde **empiezan** y **terminan**.

## ¿Y qué carácter uso para la indentación?

El que quieras.

Vale:

- Dos espacios en blanco
- Tres espacios en blanco
- Cuatro espacios en blanco
- Una tabulación

- ...

Lo **importante** es que tu estilo de *indentación* sea **coherente**. El intérprete Python se encargará de entenderlo correctamente.

## La práctica hace al maestro

La *indentación* en Python comienza siendo algo extraña y anti-intuitiva. Hace falta practicar para perderle el miedo.

**Añadamos** un nuevo fichero de código `functions.py` a la carpeta `intro/`. Puedes hacerlo con *click* derecho > New > Python File.

Escribiremos una función sencilla que **no haga nada**:

```
def do_nothing():
```

¡Un momento! Eso **falla**.

PyCharm lo subraya en rojo, y si pasamos el ratón por encima indica `Indent expected`.

**pass**

Precisamente, como Python se basa en *indentaciones* para comprender *qué* es parte de una función... ¡Hay problemas si la función está vacía!

Para este caso particular, existe la palabra reservada `pass`:

```
def do_nothing():  
    pass
```

¡Primera función completada!

## Una segunda función

Probemos ahora a hacer algo. Como sólo sabemos hacer un `print` ... Hagámoslo:

```
def do_something():  
    print("I am doing something")
```

## ¿Y devolver un valor?

Antes hemos comentado que **no** se *declara* el tipo de dato devuelto.

Pero **sí** podemos **devolver** un valor. Se usa `return` acompañado de la variable que queramos:

```
def return_something():  
    a_number = 91  
    return a_number
```

## Una peculiaridad

Nótese que en los dos primeras funciones, donde *no* devolvíamos nada, se considera que devolvemos `None`.

`None` es el equivalente en Python para `null` en Java.

Digamos que:

```
def this_is_python_code():  
    print("Believe it!")
```

...y:

```
def this_is_python_code():  
    print("Believe it!")  
    return None
```

...serían **equivalentes**.

## La tarea

**Se pide** que en tu fichero `functions.py` las tres funciones vistas arriba. Si es así, puedes verificar que están bien mediante el fichero `test_002functions.py`

## Por último

Haz `commit` y `push` para subir los cambios al repositorio.

---



Rubén Montero @ruben.montero changed milestone to [%Sprint 3](#) 4 days ago