


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

## Varios ficheros

### Resumen

- Veremos cómo funciona la sentencia `import` en Python
- Crearemos un fichero nuevo en `intro/`
- Importaremos `functions` e invocaremos funciones

### Descripción

Nuestras funciones en `functions.py` son muy bonitas, pero **no** las estamos *invocando*.

Es decir, si ejecutamos directamente:

```
python functions.py
```

¡No se imprime nada por pantalla!

Aunque los *tests* las usan, pero... ¡Nosotros **no**!

#### `import`

Mientras que en Java la sentencia `import` se usaba para *importar* una librería y poder usarla... En Python sirve para **dos** cosas:

- Importar una librería (también llamada *paquete*)
- Importar un fichero (también llamado *módulo*)

Puedes [leer algo más](#) sobre la nomenclatura y cuestiones relevantes a los `import` en Python. Nosotros, de momento, sólo consideramos lo siguiente:

- Si importamos una librería esencial, el `import` funciona **siempre** (e.g.: `import sys`)
- Si importamos una librería instalable, el `import` funciona si la librería está *instalada* en nuestro sistema
  - Para instalar nuevos paquetes se suele emplear `pip`, por ejemplo, `pip install bcrypt`. Ya lo veremos
  - Los entornos virtuales tienen por objetivo ahorrarnos problemas de dependencias
- Si importamos un fichero, debe ubicarse en la carpeta actual.
  - Pueden importarse de otras ubicaciones; aunque no profundizaremos

### ¡Práctica, por favor!

**Añadamos** un nuevo fichero `simple_program.py` a `intro/`.

Como primera línea, escribamos:

```
import functions
```

Ahora, ya **podemos** *invocar* las funciones del otro fichero.

```
import functions

functions.do_something()
```

### ¿Hay que escribir el nombre como *prefijo*?

Has notado que no *invocamos* `do_something` directamente, como haríamos en Java. Aquí usamos el *nombre del módulo* seguido de un punto (`.`).

Esto es típico de Python y **da solución a problemas de espacios de nombres**. Es decir, **no importa** que **dos** ficheros definan la **misma** función: Como hay que poner el prefijo, no hay posibilidad de conflicto.

**Pero es tedioso, ¿no?**

Sí.

Para que nuestro caso sencillo sea más ágil, podemos estrenar otro tipo de sentencias: `from {...} import {...}`.

Por ejemplo:

```
import functions
from functions import do_nothing

functions.do_something()
do_nothing() # ¡Ya no lleva prefijo!
```

## La tarea

**Se pide** que, en `functions.py` añadas una nueva función:

- Se llamará `get_color`
- No recibirá parámetros
- Únicamente, hará un `return` del string **"Amarillo"**

Además, **se pide** que, en `simple_program.py` importes la función anterior (usando el método que prefieras) y:

1. La *invoques* para asignar su resultado a una variable
2. Hagas `print("Ahora, vemos el color:")`
3. Añadas otro `print` con el valor de la variable. O sea, el color

**Puedes** borrar el resto de código de `simple_program.py`

**Puedes** probar a ejecutar `simple_program.py` y ver que se imprime por consola el resultado esperable

## Por último

Verifica que tu código pasa el *test* asociado a la tarea correctamente.

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



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