


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

# Importar tiene consecuencias

## Resumen

- Hablaremos de *dónde* se puede escribir código Python: En cualquier lado
- Comprenderemos que un `import` *ejecuta* el fichero importado
- Veremos la sentencia `if __name__ == "__main__"` como solución para evitar efectos indeseados
- Acompañaremos estos ejemplos de los ficheros `greeter.py` y `other_program.py`

## Descripción

Seguramente te habrás dado cuenta que podemos escribir código Python mucho más libremente que en Java. Como *cualquier* fichero puede ser el *punto de entrada* de una aplicación... ¡Parece que no hay límites!

**Tampoco es necesario** *definir funciones* en **ficheros separados**, como hicimos en las tareas anteriores.

Por ejemplo, en nuestro `helloworld.py`:

```
message = 'Hello, world!'
print(message)
```

Podríamos definir una función y usarla, dentro del **mismo** fichero:

```
def how_are_you()
    return "How are you?"

message = 'Hello, world!'
print(message)

other_message = how_are_you()
print(other_message)
```

## Un gran poder conlleva una gran responsabilidad

Spiderman ya nos advirtió, y debemos ser consecuentes.

Escribir libremente mucho código esparcido y desordenado en varios ficheros, provocará **problemas** tarde o temprano.

En especial por una razón:

## La sentencia `import` ejecuta un fichero

En Java, la *información* de clases y métodos se encuentra **compilada**. Al hacer `import`, *traemos* dicha *información*, sin *ejecutarla*.

Pero, recordemos que Python es **interpretado**. Por eso, para *importar* un fichero, Python lo **interpreta**. Es decir, lo **lee línea a línea**. Y para el intérprete, **leer** es **ejecutar**.

## Un ejemplo

Añadiremos un nuevo fichero `greeter.py` a `intro/`.

Declararemos inicialmente una función que devuelve un `string`:

```
def good_morning():
    return "Good morning! I hope you enjoy your day!"
```

Y luego, como en el ejemplo de arriba, podemos *invocar* la función:

```
def good_morning():
    return "Good morning! I hope you enjoy your day!"

message = good_morning()
```

```
print(message)
```

(Por cierto, podemos ahorrarnos una línea)

```
def good_morning():  
    return "Good morning! I hope you enjoy your day!"  
  
print(good_morning())
```

¡Muy bien! Al ejecutar este fichero desde el terminal (cmd.exe) con `python greeter.py`, o con **Run** en PyCharm, veremos la salida esperada:

```
"Good morning! I hope you enjoy your day!"
```

## Otro fichero

Añadamos un `other_program.py`:

```
print("I wish I could say good morning...")
```

¿Y si queremos **usar** la función `good_morning` desde otro fichero?

Un momento, ¡podemos hacerlo!

## La sentencia `import` que ya conocemos

Basta con que nuestro `other_program.py` sea:

```
from greeter import good_morning  
  
print("I wish I could say good morning...")  
print("Wait, I can!")  
  
my_variable = good_morning()  
print(my_variable)
```

Probamos a **ejecutar** `other_program.py` y...

## Ups... ¿Qué pasa?

Verás que tu `other_program.py` no funciona como es debido.

La salida producida es:

```
Good morning! I hope you enjoy your day!  
I wish I could say good morning...  
Wait, I can!  
Good morning! I hope you enjoy your day!
```

¿**Por qué** sale el mensaje `Good morning! I hope you enjoy your day!` **dos** veces?

Uhm... 🤔

Como hemos dicho, cuando se *importa* un fichero, se **lee** línea a línea.

Podríamos decir que:

```
from greeter import good_morning  
  
# FICHERO DOS (other_program.py)  
print("I wish I could say good morning...")  
print("Wait, I can!")  
  
my_variable = good_morning()  
print(my_variable)
```

...es equivalente a:

```
# FICHERO UNO (greeter.py)
```

```
def good_morning():
    return "Good morning! I hope you enjoy your day!"

print(good_morning())

# FICHERO DOS (other_program.py)
print("I wish I could say good morning...")
print("Wait, I can!")

my_variable = good_morning()
print(my_variable)
```

Vale, al importar un fichero se ejecuta...

Sí.

¿Y las funciones se invocan?

No, las funciones **no** se invocan.

Sólo se **ejecuta** el código en el *primer nivel de indentación*.

¿Siempre pasa?

Sí.

¿Y no puedo *importar* un fichero sin *ejecutarlo*?

Puedes **evitar** escribir código en el *primer nivel de indentación*.

Si tu fichero **debe** lanzar código cuando es *ejecutado*, dicho código deberá ir **englobado** en un **if** como este:

```
if __name__ == "__main__":
    # Aquí el código
```

¿Eso qué es?

Un **if** que contiene variables especiales. En **resumen**, es útil porque:

- El código dentro de `if __name__ == "__main__":`
  - Se **ejecuta** si lanzamos el fichero Python directamente (desde el terminal o PyCharm)
  - **No** se **ejecuta** si el fichero es **importado**

Arreglando nuestro **greeter.py**

Por lo tanto, **greeter.py** debería ser así:

```
def good_morning():
    return "Good morning! I hope you enjoy your day!"

if __name__ == "__main__":
    print(good_morning())
```

Puedes verificar que:

- Si ejecutas **greeter.py**, se imprime:

```
Good morning! I hope you enjoy your day!
```

- Si ejecutas **other\_program.py**, se imprime:

```
I wish I could say good morning...
Wait, I can!
Good morning! I hope you enjoy your day!
```

(Ya no hay un **print** duplicado)

## Conclusión

---

A partir de ahora, **no** escribiremos código **directamente** en el primer nivel de los archivos `.py`.

Si nuestro fichero Python, aparte de definiciones de funciones o clases, **contiene código que queremos lanzar**, debemos escribirlo **dentro de**

```
if __name__ == "__main__":
```

(Nota: No cambies `helloWorld.py`)

## Por último

Si seguiste los pasos de esta tarea y tienes `greeter.py` y `other_program.py`, tu código pasará el `test` asociado a la tarea correctamente. Compruébalo.

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

---



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