

## Ejemplo de modularización

Se va a crear el script `saludo.py` que me permitirá saludar a alguien, de hecho se podría probar su funcionamiento ejecutando la función dentro del mismo código como se muestra a continuación:

```
def saludar(nombre):  
    print(f'Hola {nombre}')
```

```
saludar('Alfonso')
```

```
Hola Alfonso
```

Coincidentemente, esta función es útil desde otro script llamado `main.py` que quiere hacer lo mismo. En este script se quiere saludar a mucha gente, por lo tanto, se hará lo siguiente:

```
import saludo as s
```

```
lista_nombres = ['Betty', 'Freddy', 'Armando', 'Marcela', 'Patricia']
```

```
for nombre in lista_nombres:  
    s.saludar(nombre)
```

Al ejecutar el programa, sorpresivamente encontramos que hay un saludo que no está en la lista ¿por qué pasa eso?

```
Hola Alfonso  
Hola Betty  
Hola Freddy  
Hola Armando  
Hola Marcela  
Hola Patricia
```

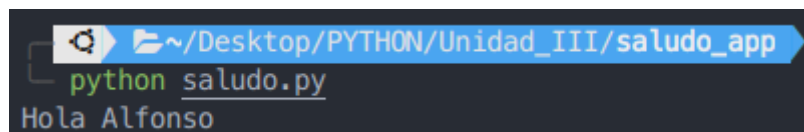
Al ejecutar la instrucción `import saludo as s`, se está ejecutando la función de `saludar()` definida en dicho script, pero eso no es un comportamiento deseado. Una solución rápida podría ser eliminar el llamado de saludar a `'Alfonso'`, pero probablemente el script `saludo.py` dejará de cumplir su objetivo que era saludar a `'Alfonso'`. Para esto se introduce el `if __name__ == '__main__':`. El propósito de este código es detectar desde dónde se está ejecutando el contenido de un script.

Python internamente tiene una variable llamada `__name__`. Esta variable guarda el nombre del módulo que se está ejecutando actualmente. Cuando uno ejecuta un script en Python, este internamente, pasa a llamarse `'__main__'`.

Para entender esto de manera práctica se ejecutará el siguiente código:

```
1 def saludar(nombre):
2     print(f'Hola {nombre}')
3
4 if __name__ == '__main__':
5     saludar('Alfonso')
```

Imagen 1. Uso `if __name__ == '__main__':`  
Fuente: Desafío Latam.



```
~/Desktop/PYTHON/Unidad_III/saludo_app
python saludo.py
Hola Alfonso
```

Imagen 2. Ejecución desde el mismo Módulo.  
Fuente: Desafío Latam.

Al ejecutar este código, Python reconoce que el código de `saludo.py` se está ejecutando. La variable interna `__name__` del script `saludo.py` pasará a llamarse `'__main__'`, lo que implica que el código `saludar('Alfonso')` se ejecutará.

¿Qué sucede si ahora ejecuto `main.py`?

```
1 import saludo as s
2
3 lista_nombres = ['Betty', 'Freddy', 'Armando', 'Marcela', 'Patricia']
4
5 for nombre in lista_nombres:
6     s.saludar(nombre)
```

Imagen 3. Código `main.py`  
Fuente: Desafío Latam.

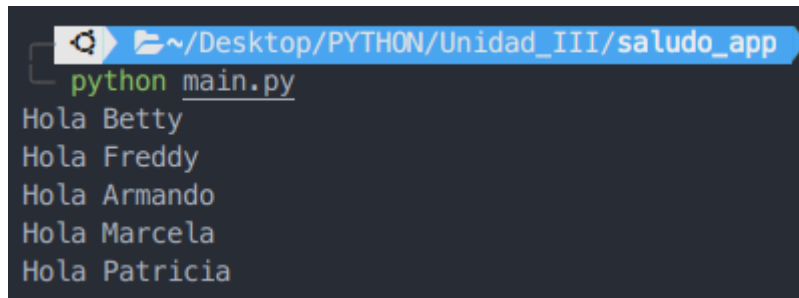
A terminal window with a dark background. The top line shows a file explorer icon and the path ~/Desktop/PYTHON/Unidad\_III/saludo\_app. Below that, the command 'python main.py' is entered. The output consists of five lines of text: 'Hola Betty', 'Hola Freddy', 'Hola Armando', 'Hola Marcela', and 'Hola Patricia'.

Imagen 4. Ejecución de `main.py` llamando con módulo corregido.  
Fuente: Desafío Latam.

Al ejecutar el programa `main.py` podemos ver que ya no se saluda a Alfonso. Esto ocurre porque Python está ejecutando `main.py`, por lo tanto la variable `__name__` de este script se llamará `'__main__'`.

Durante la ejecución de `main.py` la línea 1 `import saludo as s` ejecutará el módulo llamado `saludo.py`. Al llegar a la línea 4 de `saludo.py` se hará la prueba lógica correspondiente, la variable `__name__` de `saludo.py` **NO SERÁ** `'__main__'`, ya que como dijimos, es `main.py` la que está utilizando el valor `'__main__'` debido a que es el script que está siendo ejecutado. Esto implica que **NO** se ejecutará el saludo a `'Alfonso'`.

Normalmente constituye una buena práctica agregar el bloque `if __name__ == '__main__':` a todo módulo no principal de un proyecto. Podríamos decir que una buena práctica es:

- Definir las funciones a utilizar en el programa
- Llamar al bloque `if __name__ == '__main__':`
- Invocar todas las funciones previamente definidas. Esto puede ser particularmente útil por ejemplo para definir algún tipo de prueba que demuestre que el código está funcionando como se espera.



**NOTA:** Cuando se crean módulos que son muy grandes, podría tomar mucho tiempo. Es por eso que Python permite la opción de compilar de manera automática los módulos en el directorio `__pycache__` bajo el nombre de `modulo.version.pyc`. Para leer más detalles acerca de esta funcionalidad puedes acceder a la documentación [acá](#).