

# **CÔNQUER BLOCKS**

# PYTHON

AMA

---

Hola @lahelen estaría bien un AMA de comprensión de listas para entenderlas mejor e ir poniendolas en práctica

Hola buenaas @lahelen para la proxima ama de python avanzado podrias explicar un poco mas lo de la recursividad de las funciones que aun no me a quedado claro de el todo. Graciaaas!

Hola @lahelen , de expresiones regulares vemos algo? Sino se puede ver en un ama ?

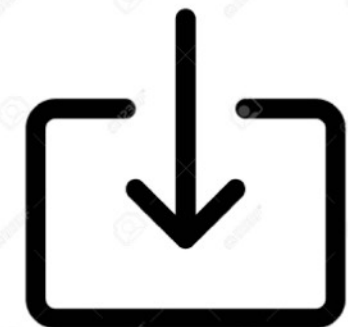
@lahelen podriamos ver en el proximo ama la construccion if name == "main":  
Es algo que no vimos pero la estuve utilizando a trabajar con diferentes al importarlos los modulos.

@lahelen podriamos ver en el proximo ama la construccion if name == "main":  
Es algo que no vimos pero la estuve utilizando a trabajar con diferentes al importarlos los  
modulos.

@lahelen podriamos ver en el proximo ama la construccion if `__name__` == "main":  
Es algo que no vimos pero la estuve utilizando a trabajar con diferentes al importarlos los  
modulos.

## ¿QUÉ ES `__name__`?

`__name__` es una *variable* especial que usamos  
al interactuar con *módulos*

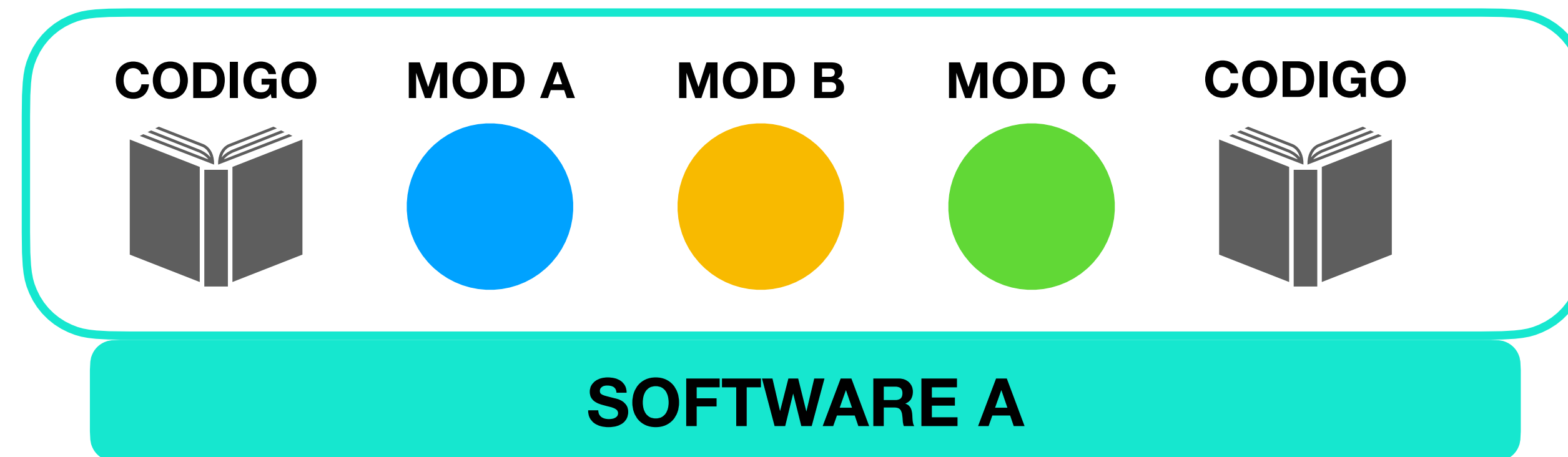


Nos ayuda a distinguir los  
módulos que han sido  
importados de los que no

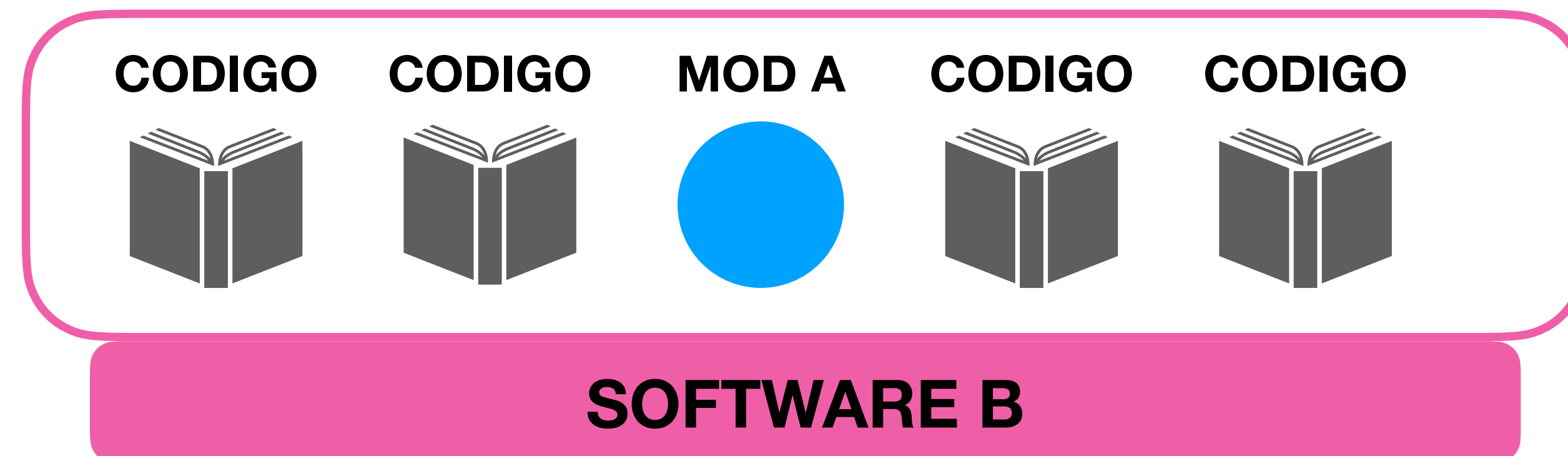


# QUE ES UN MÓDULO...

Trozo de código *autocontenido*,  
*separable* del resto del código



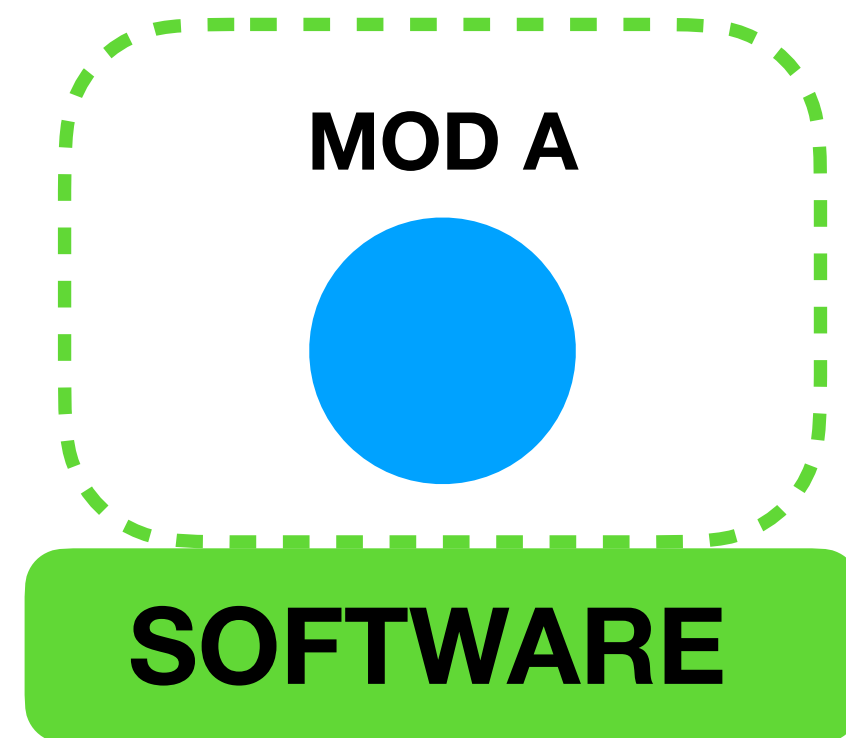
Un modulo de un software puede usarse en otro software



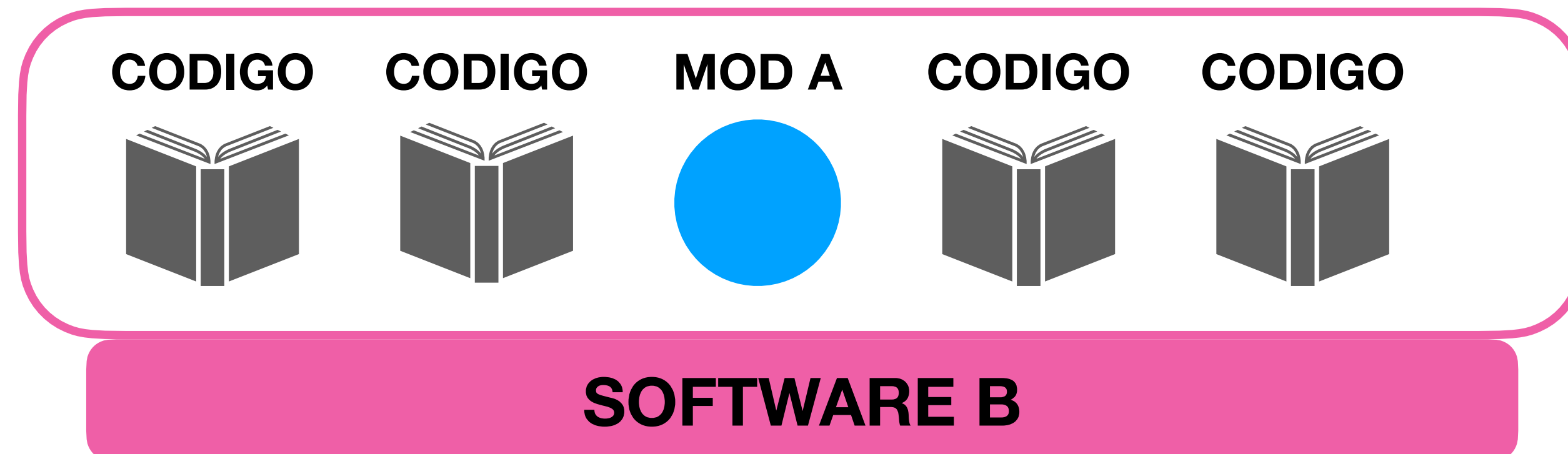


# QUE ES UN MÓDULO...

Trozo de código *autocontenido*,  
*separable* del resto del código



Un modulo también puede  
ejecutarse de manera aislada



# QUE ES UN MÓDULO...

Trozo de código *autocontenido*,  
*separable* del resto del código

Será un módulo todo aquello que...

1. se pueda re-utilizar en un software diferente
2. se pueda ejecutar de manera independiente
3. sea identificable entre distintas unidades

Por lo tanto todos los siguientes elementos son *módulos*



Librerías



Funciones



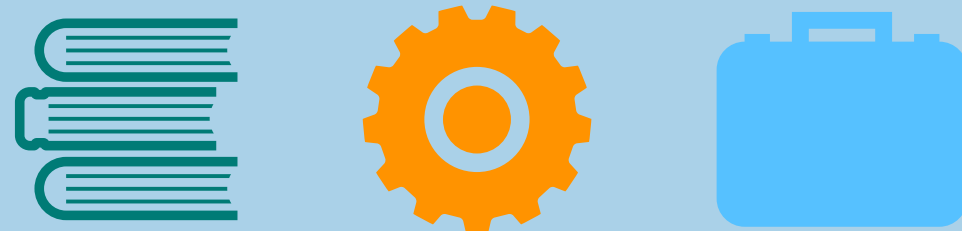
Clases



Archivo

**\_\_name\_\_ nos ayuda a distinguir los módulos que han sido importados de los que no**

**A.**    librerías  
funciones  
clases




**que importamos a  
nuestro código**

**B.**    el archivo o código  
de Python que  
estamos ejecutando  
en nuestra consola



**donde estamos  
importando las cosas**



 mi\_script.py ×

Top Level Code

if\_name\_main >  mi\_script.py

1 print("esto es un modulo")

PROBLEMAS

SALIDA

CONSOLA DE DEPURACIÓN

TERMINAL

- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$ python mi\_script.py  
esto es un modulo
- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$ █

 mi\_script.py ×

Top Level Code

if\_name\_main >  mi\_script.py > {} numpy

1 `import numpy`

2 `arr1 = numpy.array([0,1,2])`

3 `print("esto es un modulo")`

 mi\_script.py ×

No es top level code

if\_name\_main >  mi\_script.py > {} numpy

1 import numpy

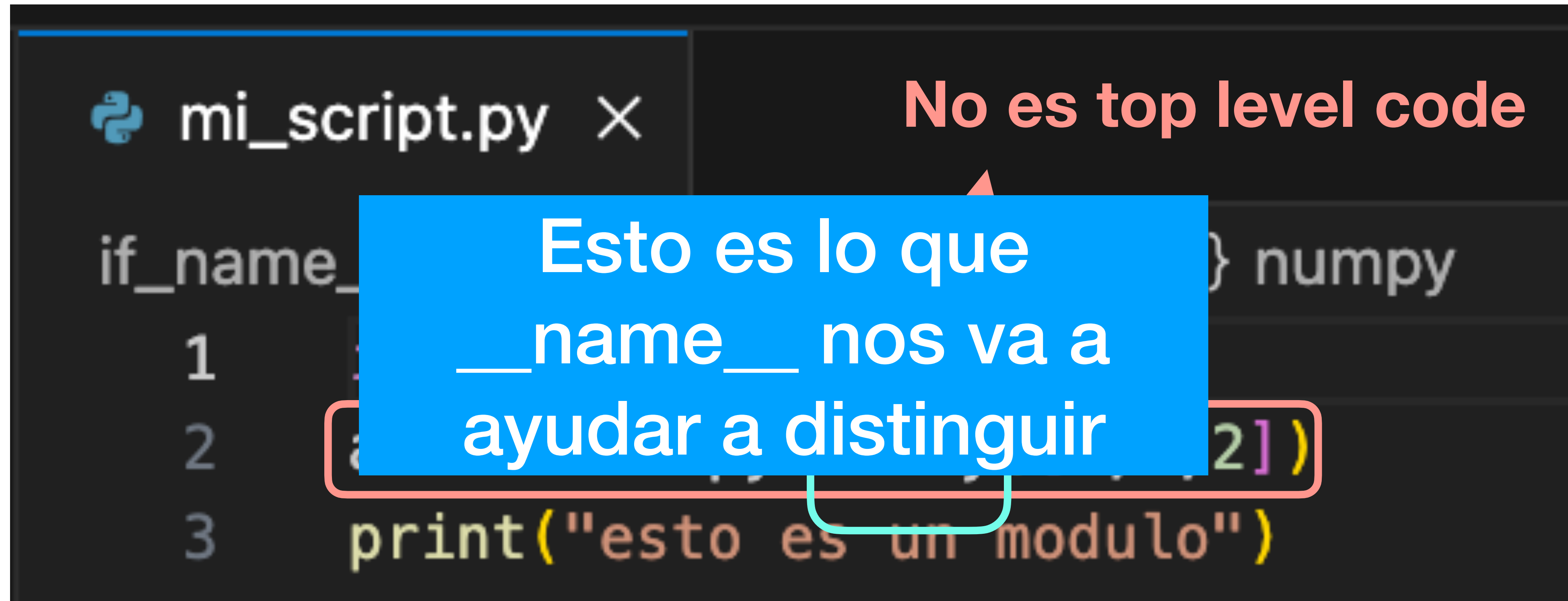
2 arr1 = numpy.array([0,1,2])

3 print("esto es un modulo")

```
mi_script.py ×  
if_name_main > mi_script.py > {} numpy  
1 import numpy  
2 arr1 = numpy.array([0,1,2])  
3 print("esto es un modulo")
```

No es top level code

Hemos tomado prestado un elemento que nunca hemos definido en nuestro código ya que pertenece a numpy



```
mi_script.py X
```

```
if_name_  
1  
2  
3 print("esto es un modulo")
```

Esto es lo que  
\_\_name\_\_ nos va a  
ayudar a distinguir

No es top level code

numpy

2]

Hemos tomado prestado un elemento que nunca hemos definido en nuestro código ya que pertenece a numpy

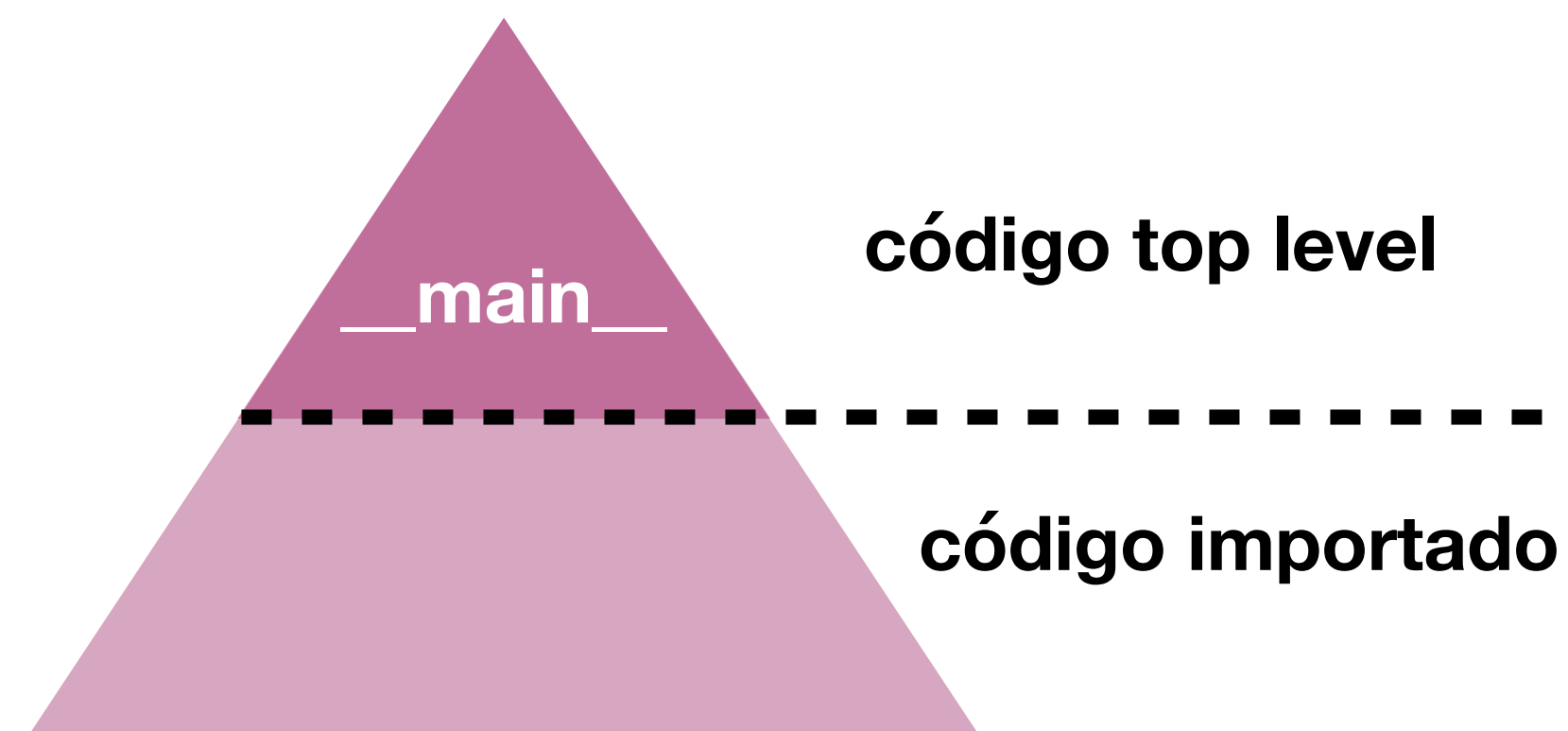


@lahelen podriamos ver en el proximo ama la construccion if `name == 'main'`:  
Es algo que no vimos pero la estuve utilizando a trabajar con diferentes al importarlos los  
modulos.

**`__main__`** representa el nombre (name) del *top level environment* donde el código está siendo ejecutado

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

estamos comprobando si el código ejecutado es top level o no



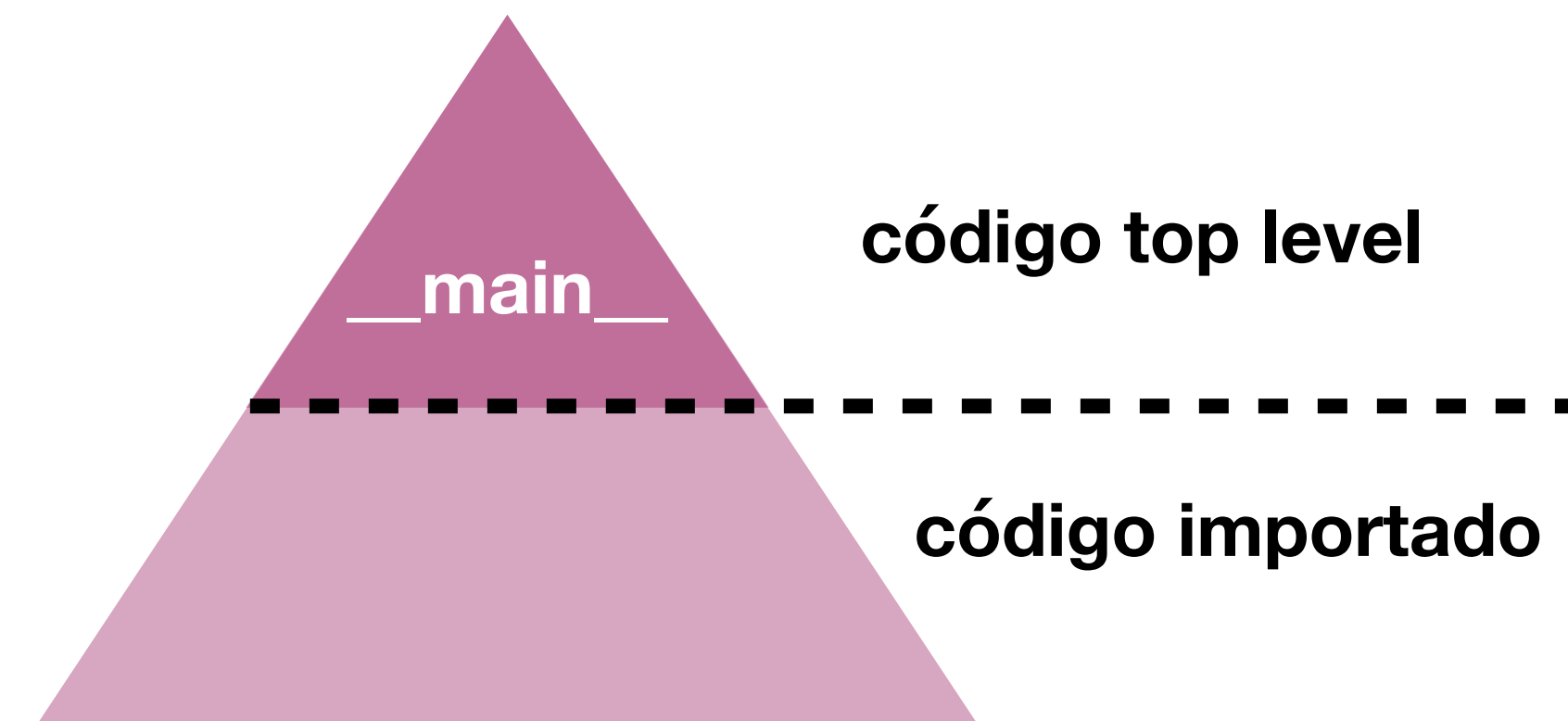


**`__main__`** representa el nombre (name) del *top level environment* donde el código está siendo ejecutado

```
mi_script.py ×  
if_name_main > mi_script.py > ...  
1 import numpy  
2 arr1 = numpy.array([0,1,2])  
3 print("esto es un modulo")  
4 print(__name__)
```

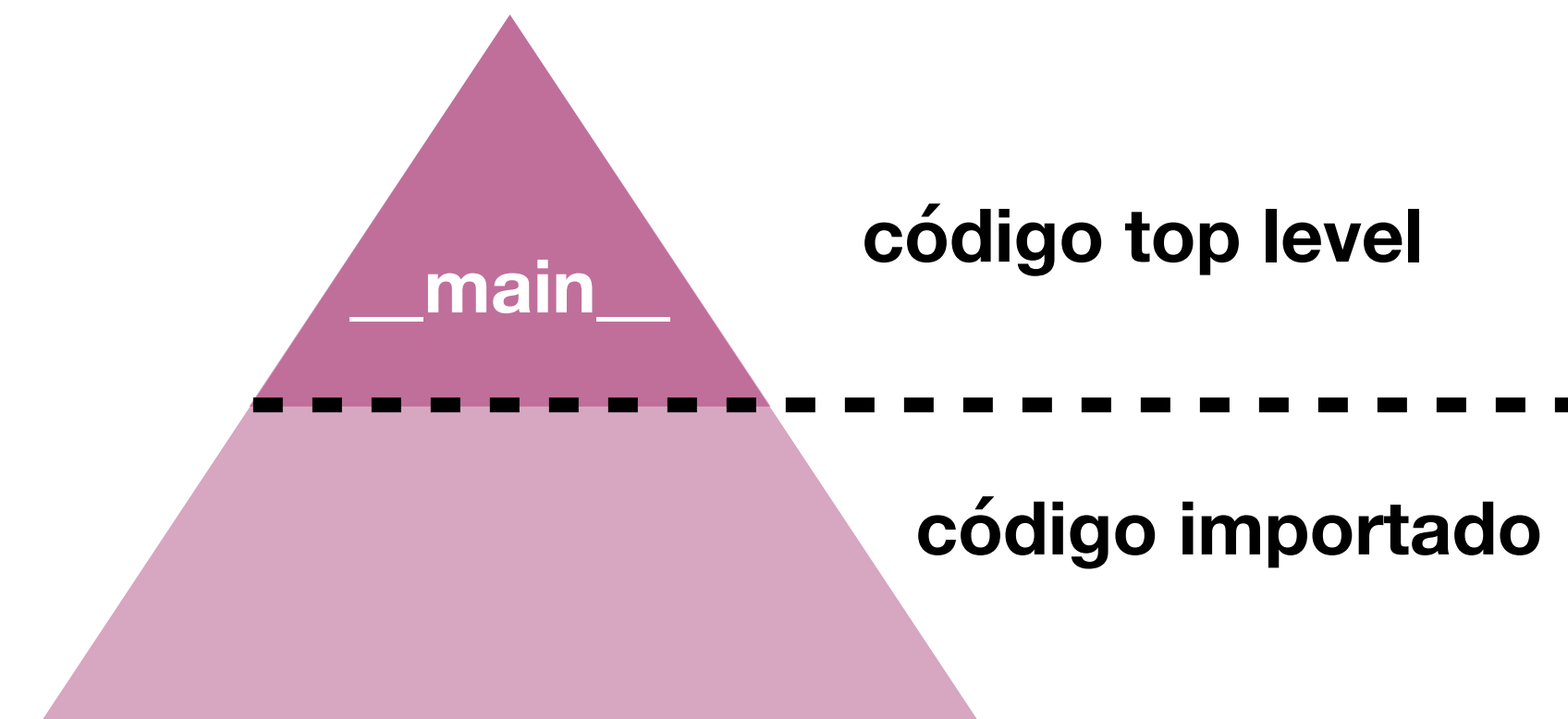
PROBLEMAS SALIDA CONSOLA DE DEPURA

● (cblocks) MacBook-Pro-5:if\_name\_main El  
esto es un modulo  
\_\_main\_\_



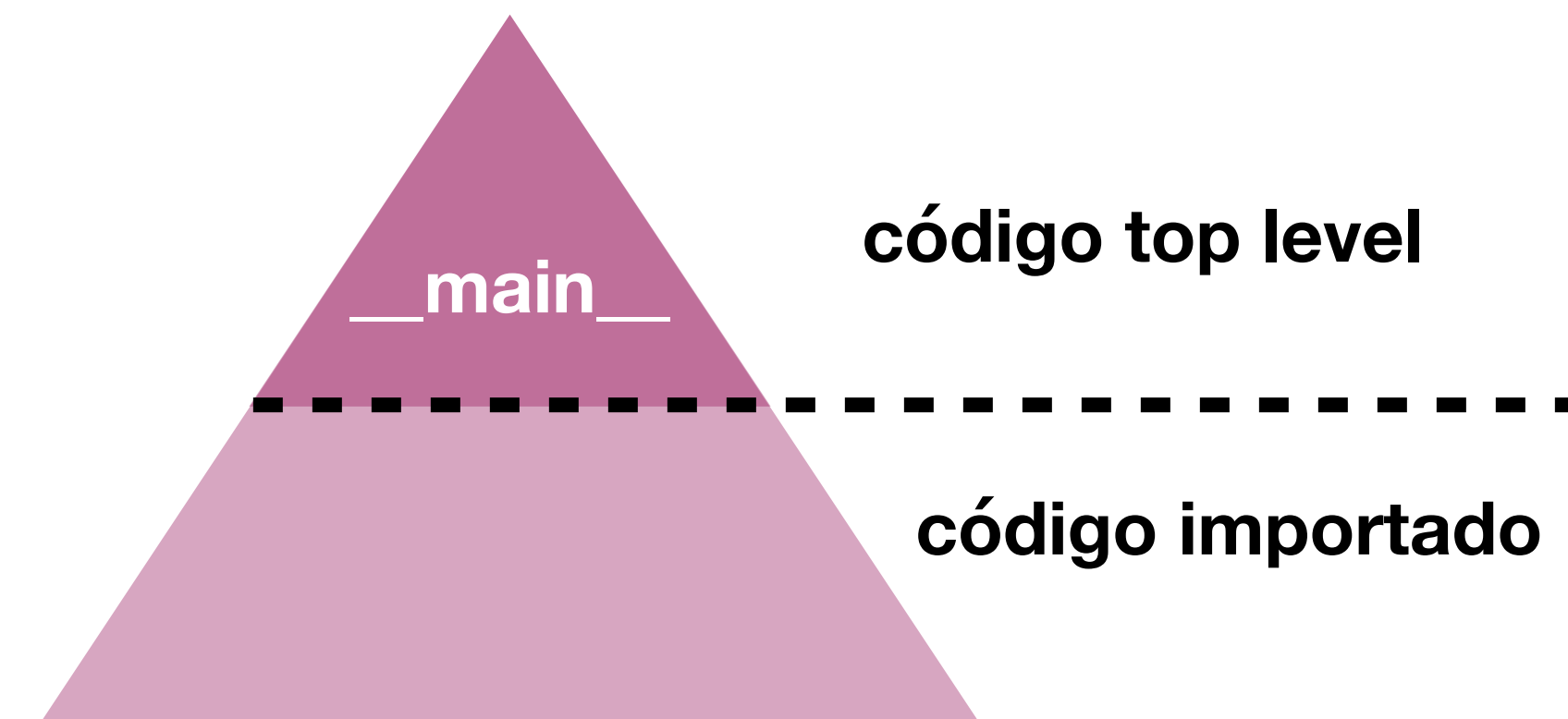
**`__main__`** representa el nombre (name) del *top level environment* donde el código está siendo ejecutado

```
mi_script.py ×  
if_name_main > mi_script.py > ...  
1 import numpy  
2 arr1 = numpy.array([0,1,2])  
3 print("esto es un modulo")  
4 print(numpy.__name__)  
  
PROBLEMAS SALIDA CONSOLA DE DEPURA  
● (cblocks) MacBook-Pro-5:if_name_main ET  
esto es un modulo  
numpy
```



**`__main__`** representa el nombre (name) del *top level environment* donde el código está siendo ejecutado

```
mi_script.py ×  
if_name_main > mi_script.py > ...  
1 import numpy as np  
2 arr1 = np.array([0,1,2])  
3 print("esto es un modulo")  
4 print(np.__name__)  
  
PROBLEMAS SALIDA CONSOLA DE DE  
  
● (cblocks) MacBook-Pro-5:if_name_mai  
esto es un modulo  
numpy  
○ (cblocks) MacBook-Pro-5:if_name_mai
```

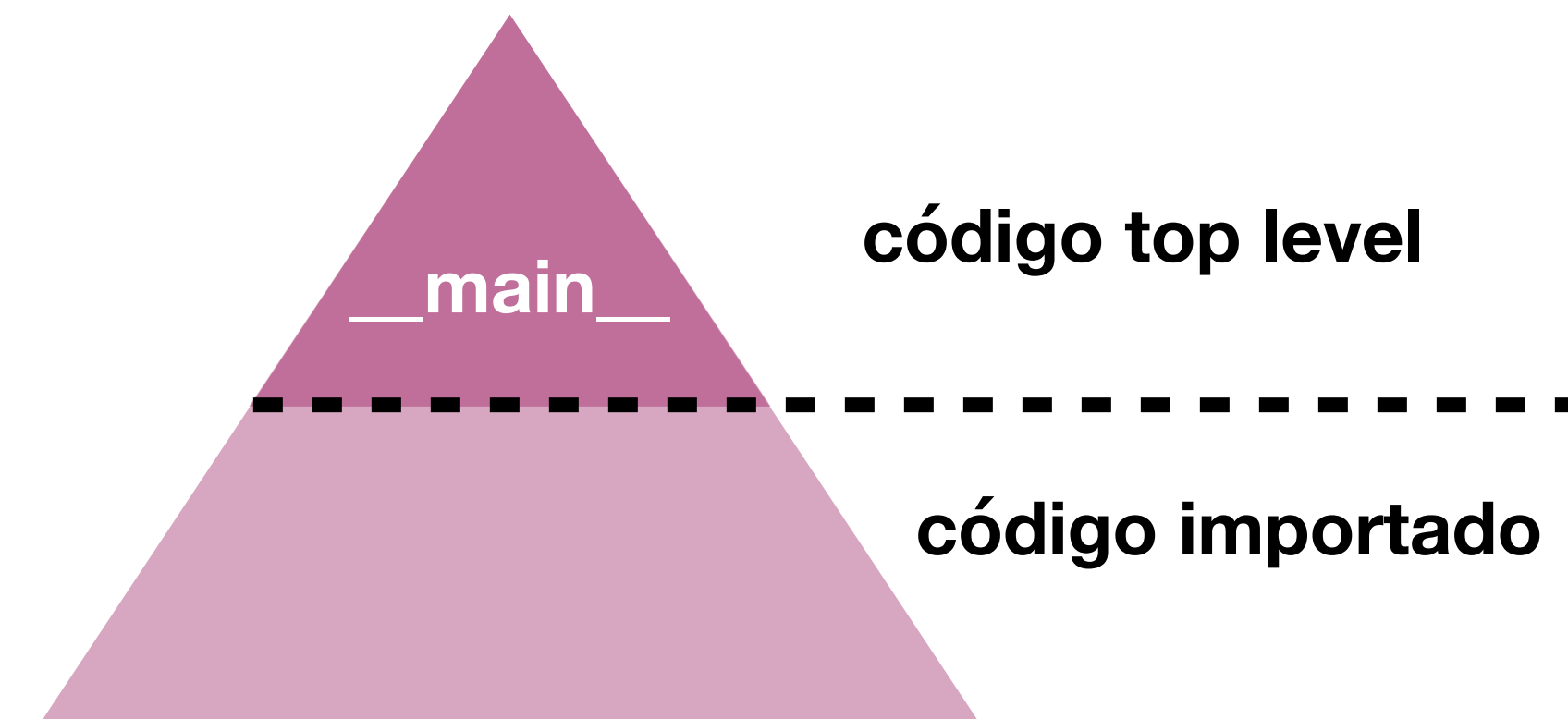


**`__main__`** representa el nombre (name) del *top level environment* donde el código está siendo ejecutado

```
mi_script.py ×
if_name_main > mi_script.py > ...
7 # y con imports específicos...
8 from datetime import timezone
9 print(timezone.__name__)
10

PROBLEMAS SALIDA CONSOLA DE DEPURAC

● (cblocks) MacBook-Pro-5:if_name_main Elen
● (cblocks) MacBook-Pro-5:if_name_main Elen
  timezone
○ (cblocks) MacBook-Pro-5:if_name_main Elen
```





**`__main__`** representa el nombre (name) del *top level environment* donde el código está siendo ejecutado

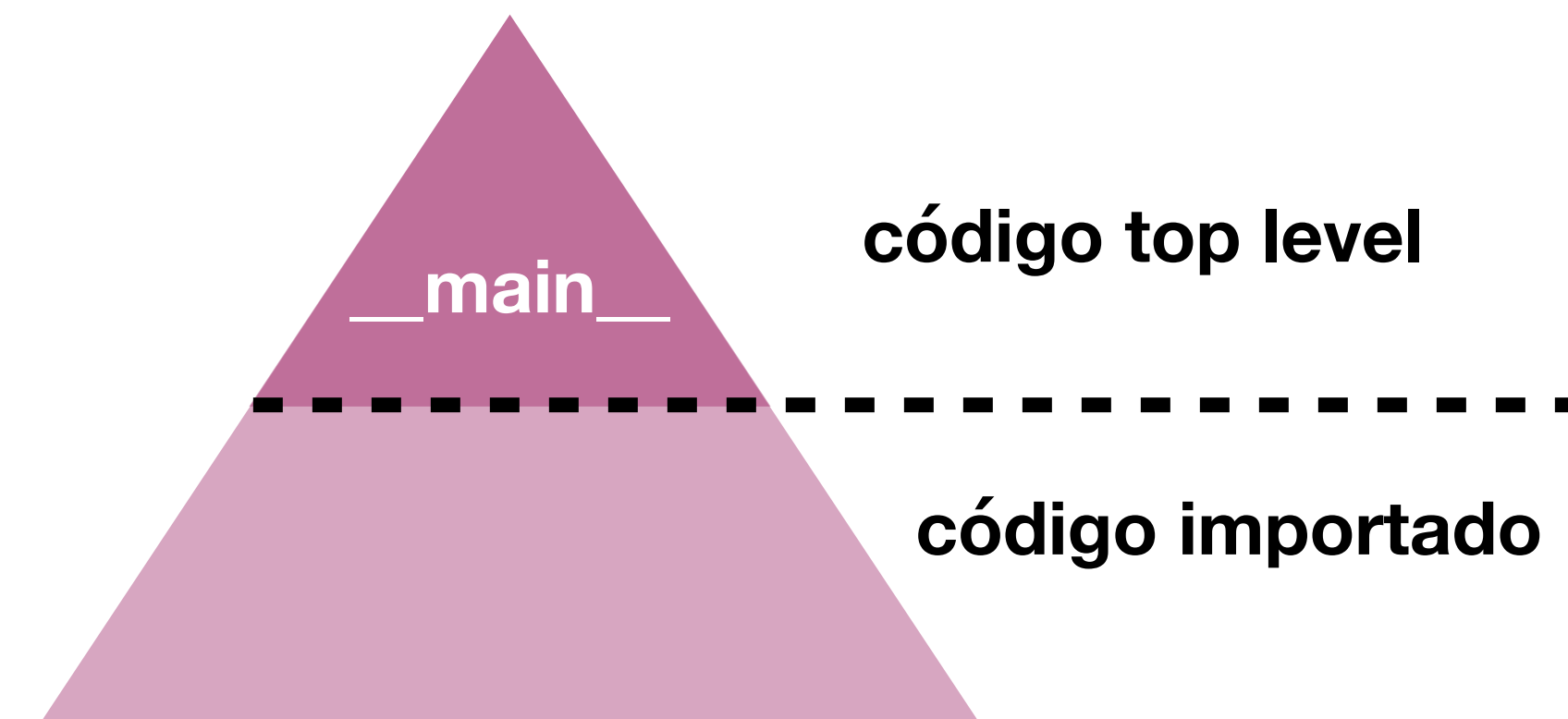
```

mi_script.py x
if_name_main > mi_script.py > ...
7  # y con imports especificos...
8  from datetime import timezone
9  print(timezone.__name__)
10 print(datetime.__name__)

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN

(cblocks) MacBook-Pro-5:if_name_main Elena$ py
timezone
Traceback (most recent call last):
  File "/Users/Elena/Desktop/Master Conquer Bl
ain/mi_script.py", line 10, in <module>
    print(datetime.__name__)
NameError: name 'datetime' is not defined
(cblocks) MacBook-Pro-5:if_name_main Elena$

```



**`__main__`** representa el nombre (name) del ***top level environment*** donde el código está siendo ejecutado

```
mi_script.py ×
if_name_main > mi_script.py > ...
7 # y con imports específicos...
8 from datetime import timezone
9 import datetime
10 print(timezone.__name__)
11 print(datetime.__name__)
12
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN

- (cblocks) MacBook-Pro-5:if\_name\_main Elena  
timezone  
datetime
- (cblocks) MacBook-Pro-5:if\_name\_main Elena

Regla general: `__name__` nos devolverá el nombre del archivo sin la extensión `.py`.

Usamos el nombre del archivo como nombre del modulo (ejemplo: *datetime*)



`__main__` representa el nombre (name) del *top level environment* donde el código está siendo ejecutado

```
mi_script.py ×  
if_name_main > mi_script.py > ...  
7 # y con imports específicos...  
8 from datetime import timezone  
9 import datetime  
10 print(timezone.__name__)  
11 print(datetime.__name__)  
12  
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN  
● (cblocks) MacBook-Pro-5:if_name_main Elena  
  timezone  
  datetime  
○ (cblocks) MacBook-Pro-5:if_name_main Elena
```

Si estamos importando una clase en específico, su nombre será el nombre de la clase (ejemplo: *timezone*)

**¿PARA QUÉ USAMOS TODO ESTO?**

```

para_importar.py ×
if_name_main > para_importar.py
1  def llamame():
2      print("¡Hola!")
3
4  llamame()
    
```

PROBLEMAS   SALIDA   CONSOLA

- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$ python3.9 para\_ejecutar.py  
¡Hola!
- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$

```

para_ejecutar.py ×
if_name_main > para_ejecutar.py > llamame
1  from para_importar import llamame
    
```

PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL

- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$ python3.9 para\_ejecutar.py  
¡Hola!
- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$

```

para_importar.py ×
if_name_main > para_importar.py
1  def llamame():
2      print("¡Hola!")
3
4  llamame()
    
```

PROBLEMAS SALIDA CONSOLA

- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$ python3.9 para\_importar.py  
¡Hola!
- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$

```

para_ejecutar.py ×
if_name_main > para_ejecutar.py > llamame
1  from para_importar import llamame
    
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$ python3.9 para\_ejecutar.py  
¡Hola!
- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$

**Vemos el output de llamame() al ejecutar el script para\_ejecutar.py, aunque en ningún momento hemos llamado a la función.**

**if \_\_name\_\_ == "\_\_main\_\_"** nos sirve para evitar esto

```

para_importar.py ×
if_name_main > para_importar.py > ..
1  def llamame():
2      print("¡Hola!")
3
4  if __name__ == "__main__":
5      llamame()
    
```

PROBLEMAS SALIDA CONSOLA DE DE

- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$ python3.9 para\_ejecutar.py
- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$
- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$

```

para_ejecutar.py ×
if_name_main > para_ejecutar.py > llamame
1  from para_importar import llamame
    
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$ python3.9 para\_ejecutar.py
- (cblocks) MacBook-Pro-5:if\_name\_main Elena\$



# **CÔNQUER BLOCKS**