

CÓNQUER BLOCKS

PYTHON

ARRAYS, MODULOS Y LIBRERÍAS

QUÉ ES UN MODULO

Módulo: Archivo con extension .py que contiene código de Python que puede importarse dentro de otro archivo de Python.

QUÉ ES UN MODULO

Módulo: Archivo con extension .py que contiene código de Python que puede importarse dentro de otro archivo de Python.



```

EXPLORADOR
└─ PRUEBA
   ├── __pycache__
   ├── area_circulo.py
   ├── constantes.py
   └─ imprimir_cte.py

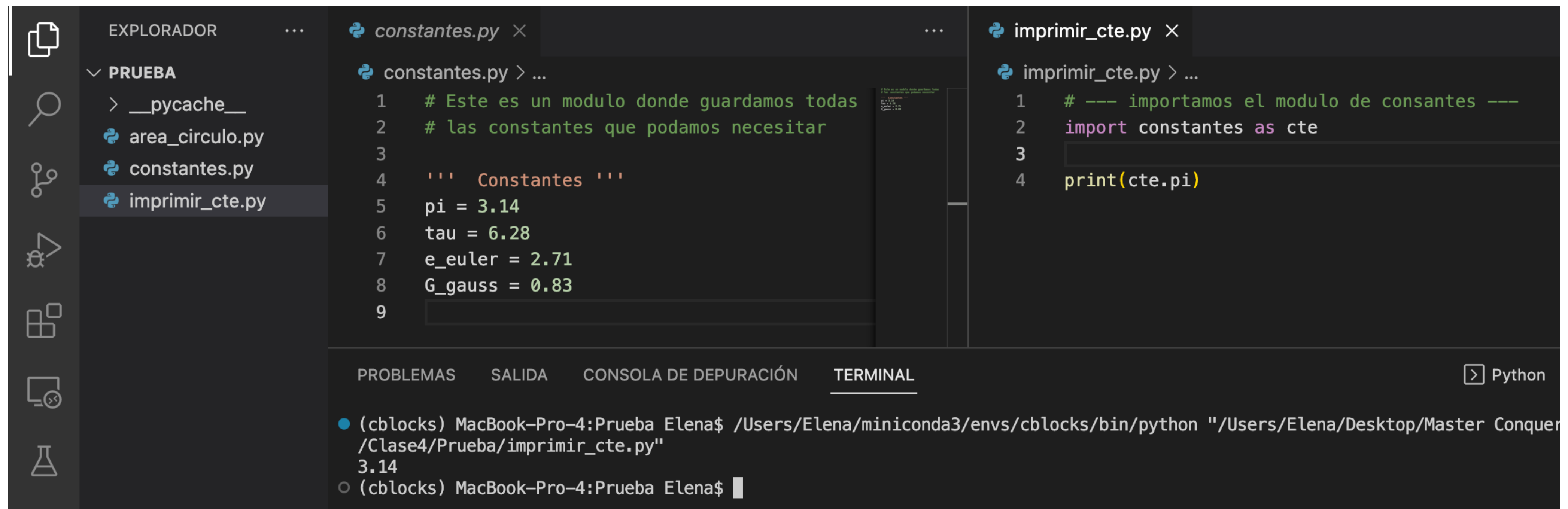
constantes.py
1 # Este es un modulo donde guardamos todas
2 # las constantes que podemos necesitar
3
4 ''' Constantes '''
5 pi = 3.14
6 tau = 6.28
7 e_euler = 2.71
8 G_gauss = 0.83
9

imprimir_cte.py
1 # --- importamos el modulo de consantes ---
2 import constantes
3
4 print(constantes.pi)

TERMINAL
(cblocks) MacBook-Pro-4:Prueba Elena$ /Users/Elena/miniconda3/envs/cblocks/bin/python "/Users/Elena/Desktop/Master Conque
/Clase4/Prueba/imprimir_cte.py"
3.14
(cblocks) MacBook-Pro-4:Prueba Elena$
  
```

QUÉ ES UN MODULO

Módulo: Archivo con extension .py que contiene código de Python que puede importarse dentro de otro archivo de Python.



```
EXPLORADOR
└─ PRUEBA
   ├── __pycache__
   ├── area_circulo.py
   ├── constantes.py
   └── imprimir_cte.py

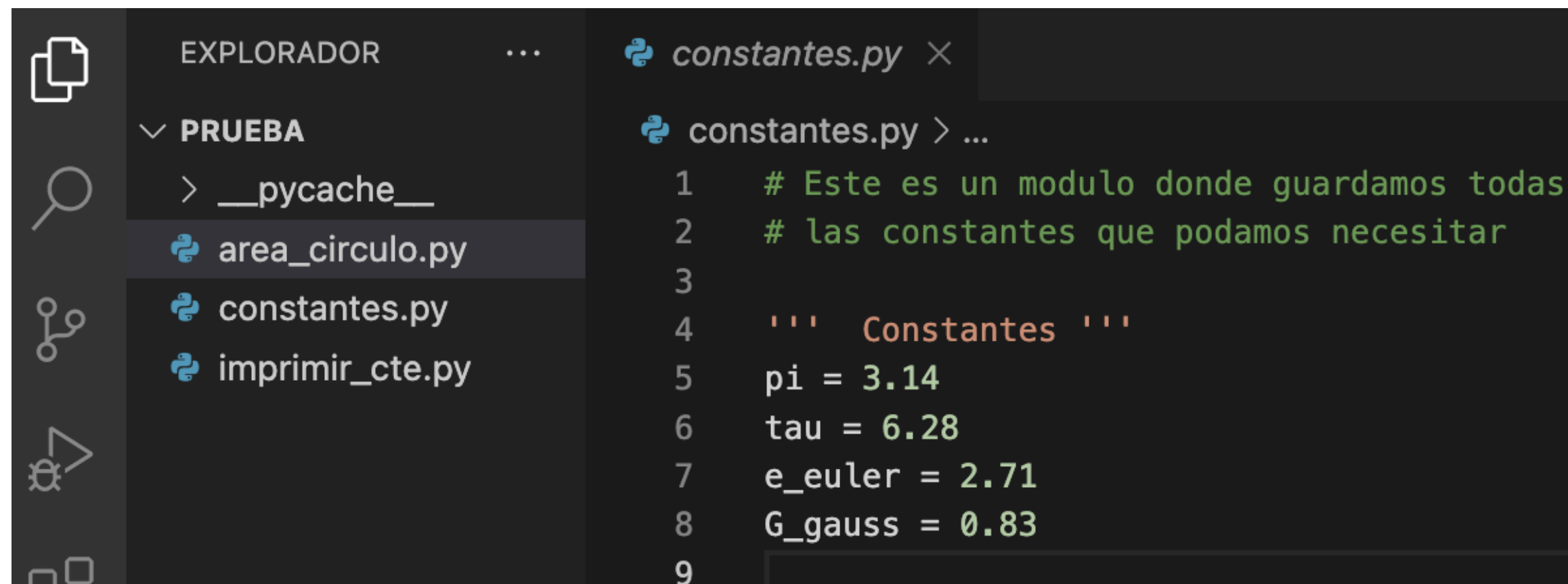
constantes.py
1  # Este es un modulo donde guardamos todas
2  # las constantes que podamos necesitar
3
4  ''' Constantes '''
5  pi = 3.14
6  tau = 6.28
7  e_euler = 2.71
8  G_gauss = 0.83
9

imprimir_cte.py
1  # --- importamos el modulo de consantes ---
2  import constantes as cte
3
4  print(cte.pi)

TERMINAL
Python
● (cblocks) MacBook-Pro-4:Prueba Elena$ /Users/Elena/miniconda3/envs/cblocks/bin/python "/Users/Elena/Desktop/Master Conquer/Clase4/Prueba/imprimir_cte.py"
3.14
○ (cblocks) MacBook-Pro-4:Prueba Elena$
```

QUÉ ES UN MODULO

Módulo: Archivo con extension .py que contiene código de Python que puede importarse dentro de otro archivo de Python.



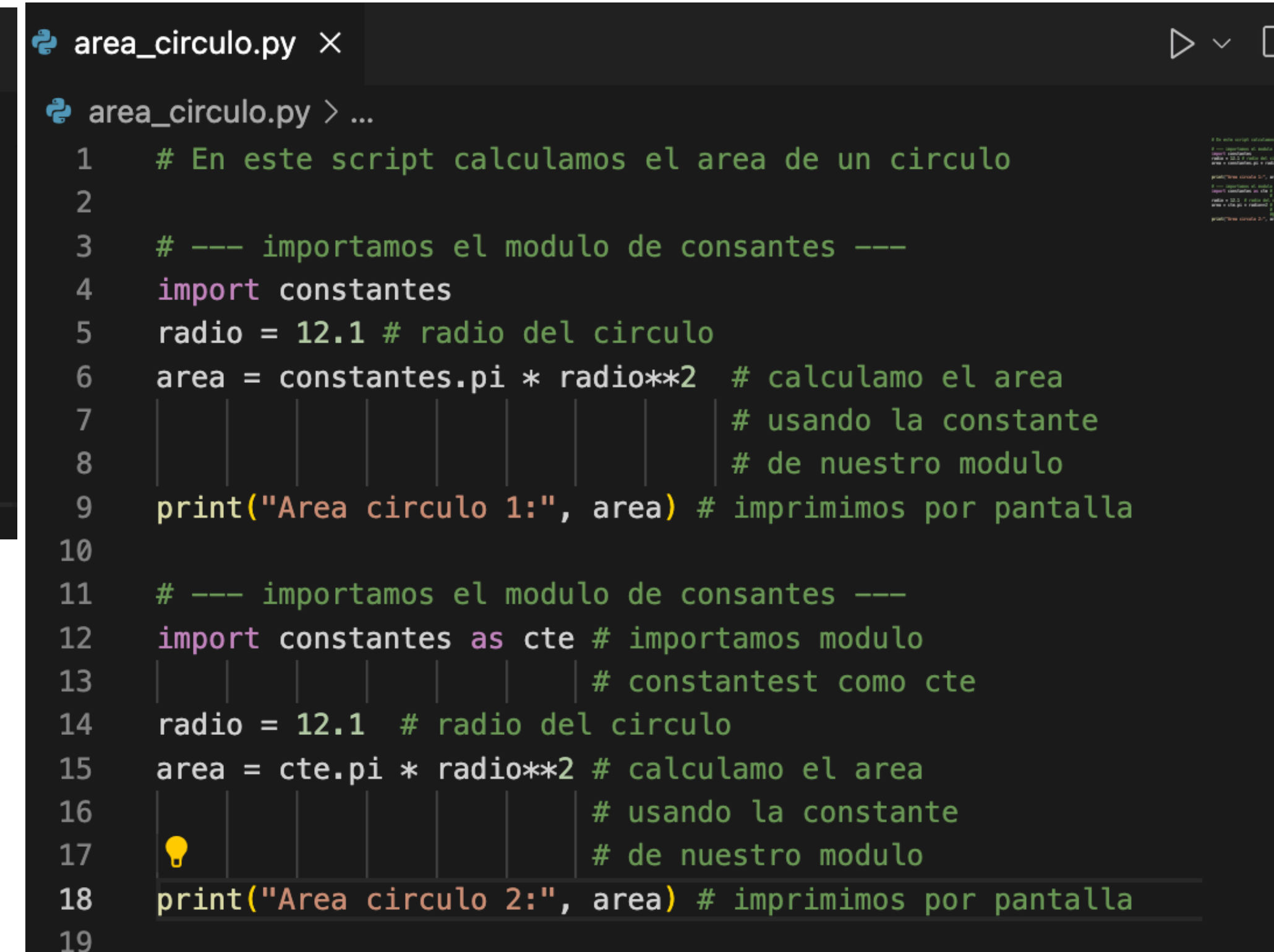
EXPLORADOR

- PRUEBA
 - __pycache__
 - area_circulo.py
 - constantes.py
 - imprimir_cte.py

constantes.py

```

1  # Este es un modulo donde guardamos todas
2  # las constantes que podamos necesitar
3
4  ''' Constantes '''
5  pi = 3.14
6  tau = 6.28
7  e_euler = 2.71
8  G_gauss = 0.83
9
    
```



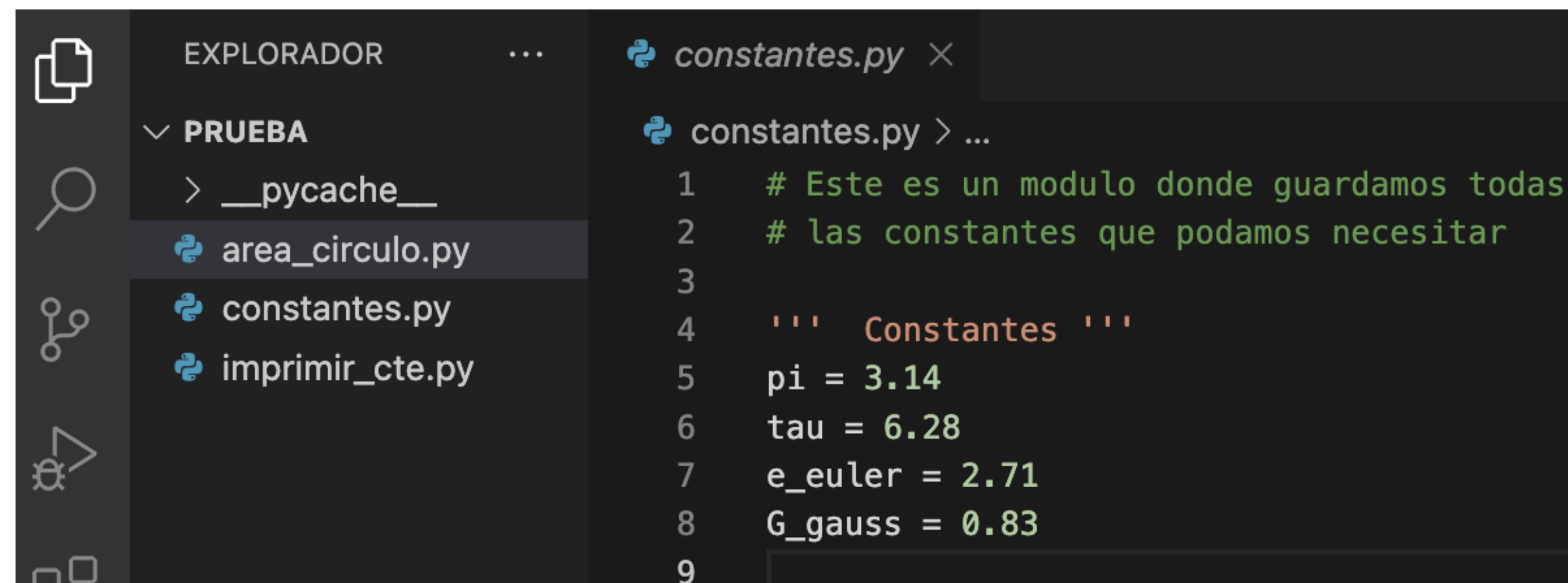
area_circulo.py

```

1  # En este script calculamos el area de un circulo
2
3  # --- importamos el modulo de consantes ---
4  import constantes
5  radio = 12.1 # radio del circulo
6  area = constantes.pi * radio**2 # calculamo el area
7  # usando la constante
8  # de nuestro modulo
9  print("Area circulo 1:", area) # imprimimos por pantalla
10
11 # --- importamos el modulo de consantes ---
12 import constantes as cte # importamos modulo
13 # constantest como cte
14 radio = 12.1 # radio del circulo
15 area = cte.pi * radio**2 # calculamo el area
16 # usando la constante
17 # de nuestro modulo
18 print("Area circulo 2:", area) # imprimimos por pantalla
19
    
```


QUÉ ES UN MODULO

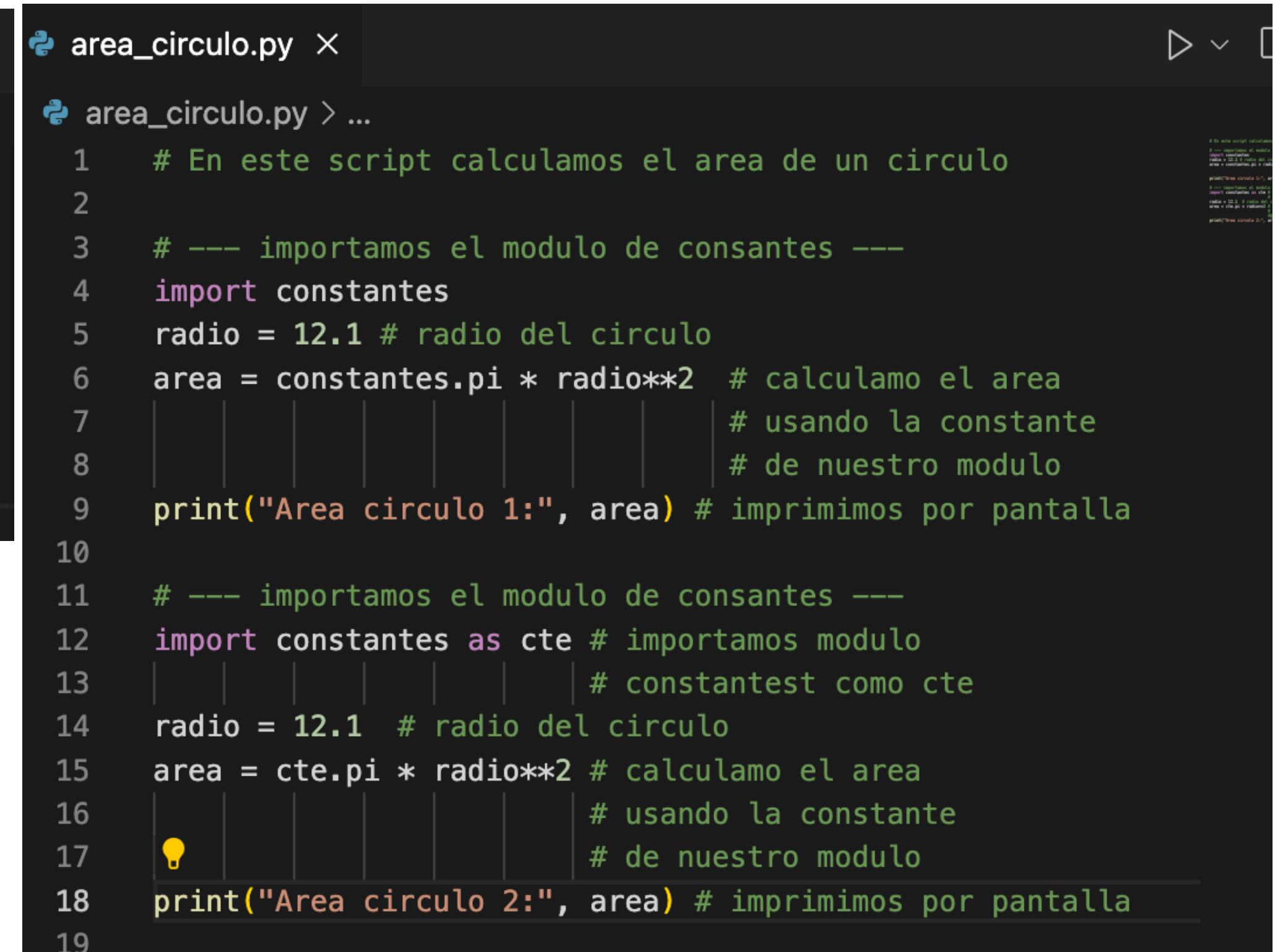
Módulo: Archivo con extension .py que contiene código de Python que puede importarse dentro de otro archivo de Python.



```

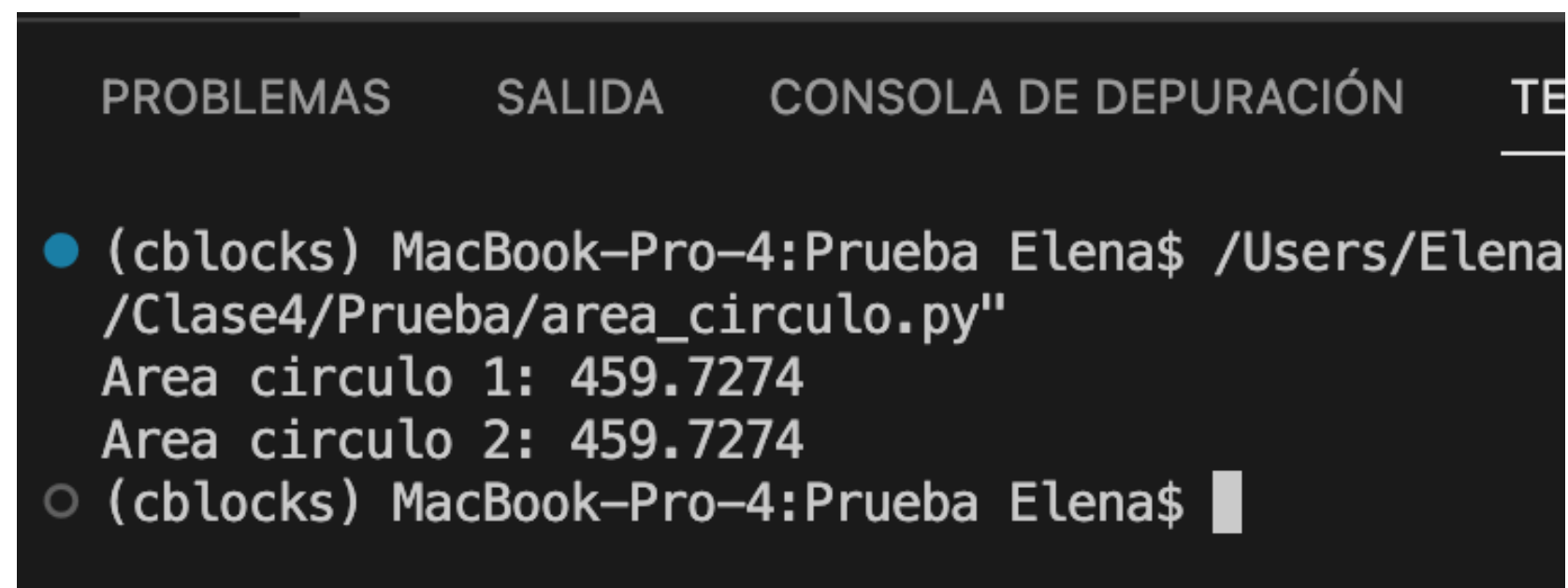
EXPLORADOR
PRUEBA
  > __pycache__
  area_circulo.py
  constantes.py
  imprimir_cte.py

constantes.py
1  # Este es un modulo donde guardamos todas
2  # las constantes que podemos necesitar
3
4  ''' Constantes '''
5  pi = 3.14
6  tau = 6.28
7  e_euler = 2.71
8  G_gauss = 0.83
9
  
```



```

area_circulo.py
1  # En este script calculamos el area de un circulo
2
3  # --- importamos el modulo de consantes ---
4  import constantes
5  radio = 12.1 # radio del circulo
6  area = constantes.pi * radio**2 # calculamo el area
7  # usando la constante
8  # de nuestro modulo
9  print("Area circulo 1:", area) # imprimimos por pantalla
10
11 # --- importamos el modulo de consantes ---
12 import constantes as cte # importamos modulo
13 # constantest como cte
14 radio = 12.1 # radio del circulo
15 area = cte.pi * radio**2 # calculamo el area
16 # usando la constante
17 # de nuestro modulo
18 print("Area circulo 2:", area) # imprimimos por pantalla
19
  
```



```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TE
• (cblocks) MacBook-Pro-4:Prueba Elena$ /Users/Elena/Clase4/Prueba/area_circulo.py
Area circulo 1: 459.7274
Area circulo 2: 459.7274
○ (cblocks) MacBook-Pro-4:Prueba Elena$
  
```

PAQUETES Y LIBRERIAS

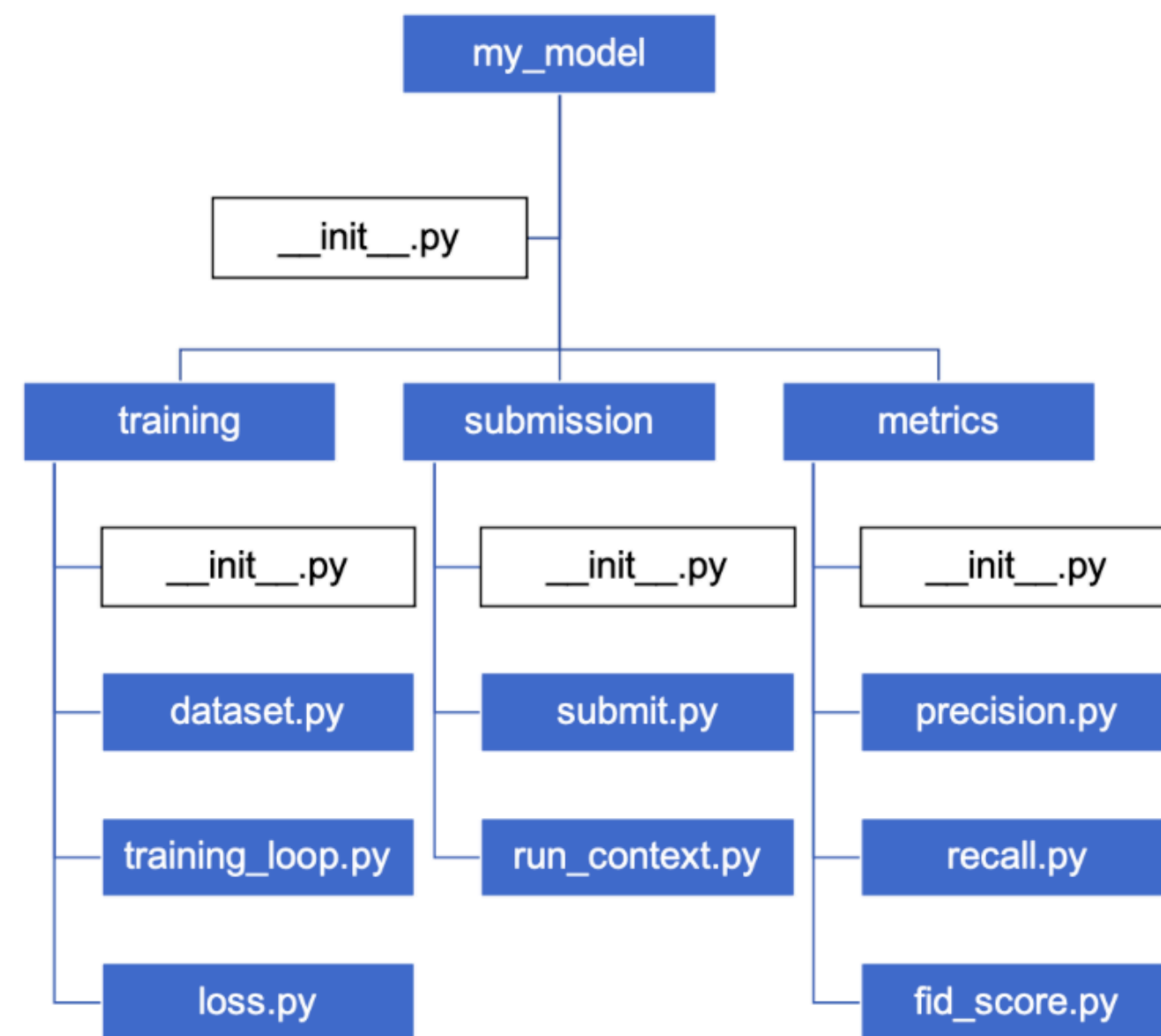
Package/Paquete: Un directorio o carpeta con una colección de módulos

Podemos tener paquetes y subpaquetes con distintos módulos en cada uno de ellos

PAQUETES Y LIBRERIAS

Package/Paquete: Un directorio o carpeta con una colección de módulos

Podemos tener paquetes y subpaquetes con distintos módulos en cada uno de ellos

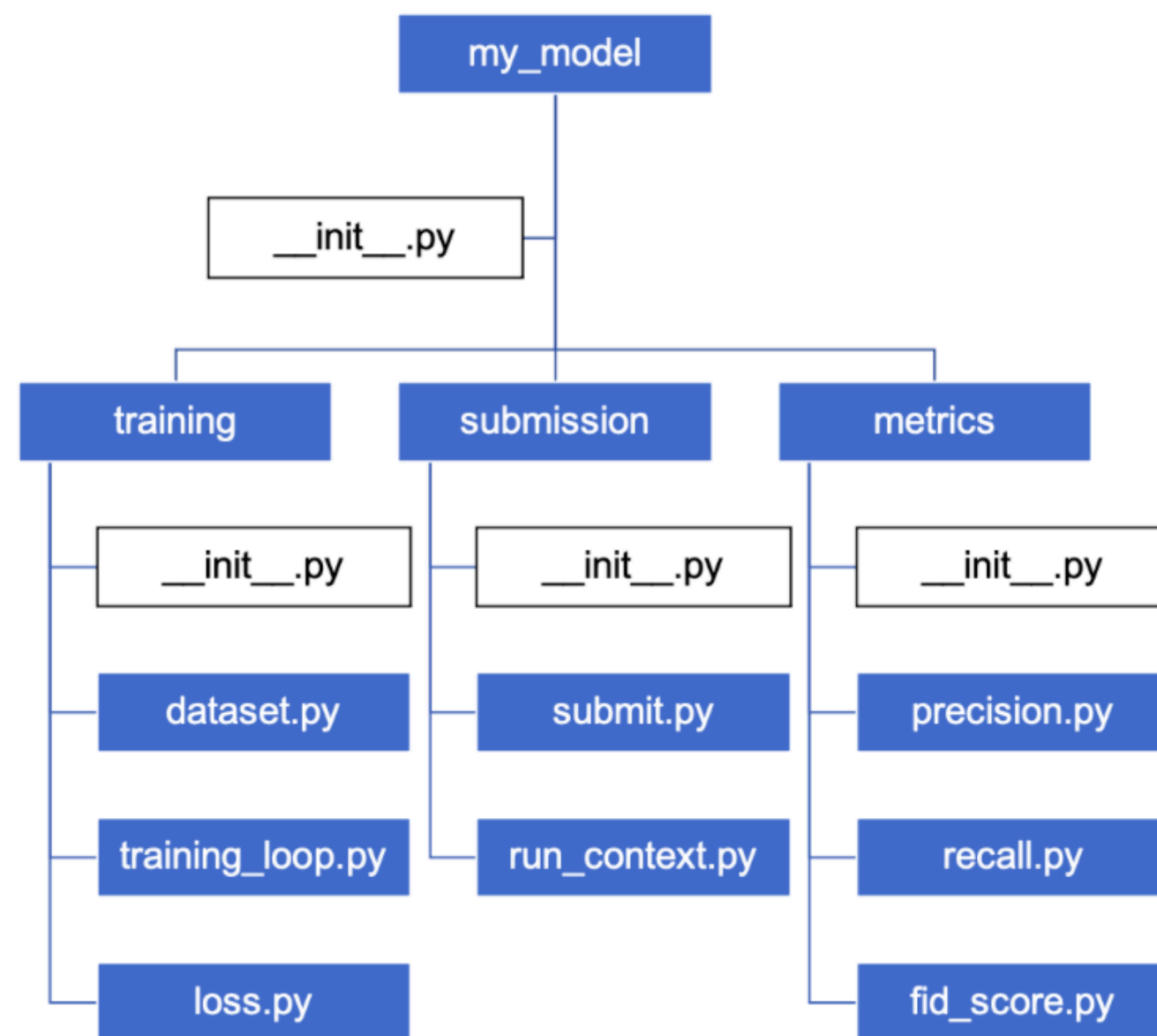




PAQUETES Y LIBRERIAS

Package/Paquete: Un directorio o carpeta con una colección de módulos

Podemos tener paquetes y subpaquetes con distintos módulos en cada uno de ellos



Library/Librería: Es un termino general para referirse a una pieza de código reutilizable.

Normalmente una librería a contiene una colección de módulos y paquetes.

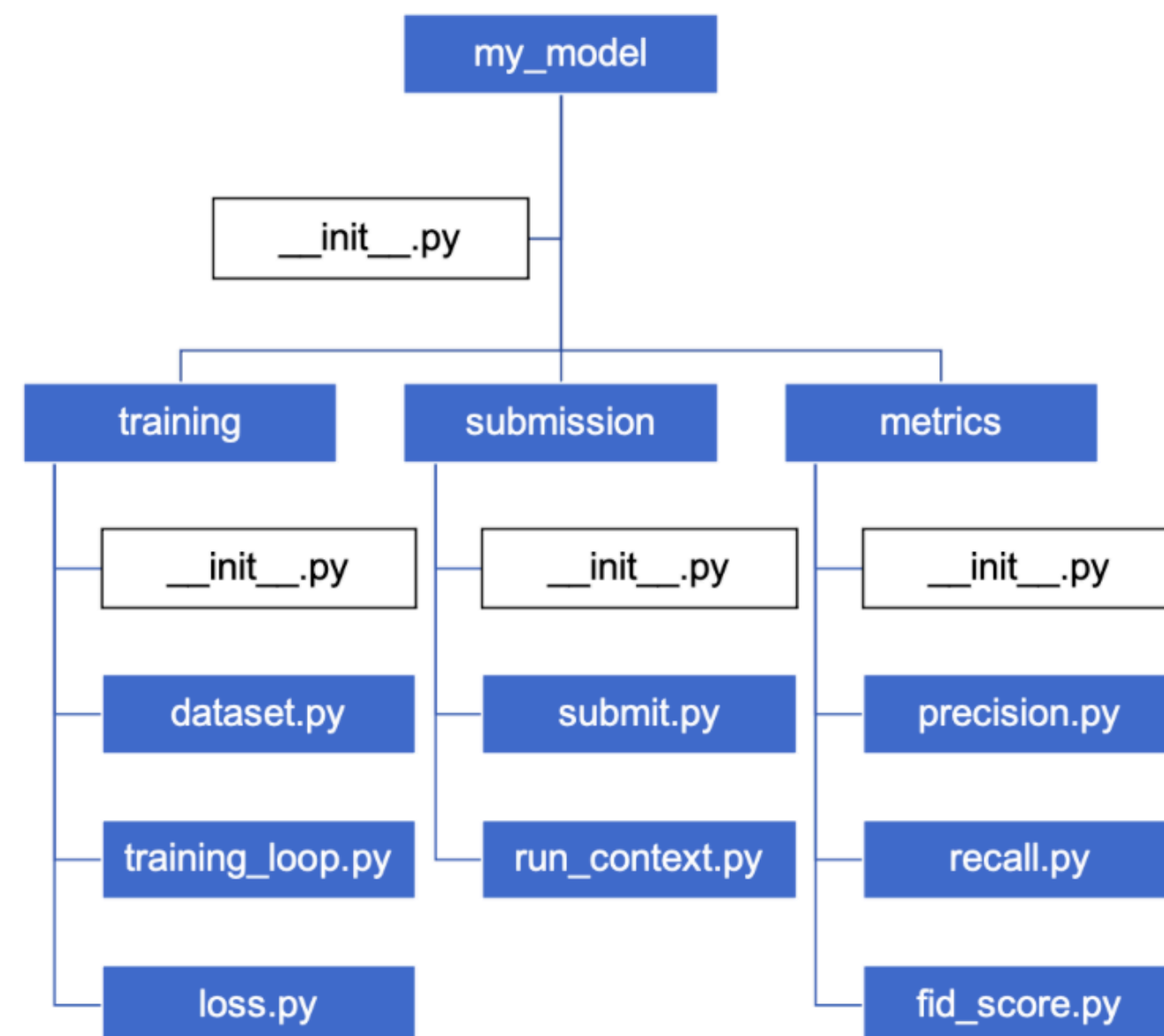
Ejemplos:

Numpy
Matplotlib
Pytorch
Pandas
...

PAQUETES Y LIBRERIAS

Package/Paquete: Un directorio o carpeta con una colección de módulos

Podemos tener paquetes y subpaquetes con distintos módulos en cada uno de ellos



Library/Librería: Es un termino general para referirse a una pieza de código reutilizable.

Normalmente una librería a contiene una colección de módulos y paquetes.

Ejemplos:

Numpy
Matplotlib
Pytorch
Pandas
...

Una librería es una colección de paquetes
Un paquete es una colección de módulos

ARRAYS

¿Qué es un Array?

Son contenedores que son capaces de guardar más de un objeto al mismo tiempo.
Colección ordenada de elementos/objetos.



ARRAYS

¿Qué es un Array? Son contenedores que son capaces de guardar más de un objeto al mismo tiempo. Colección ordenada de elementos/objetos.

Lista	Array
Es una estructura in-built	Es una estructura que es necesario importar usando <i>array</i> o <i>numpy</i>
Se crea usando corchetes []	Se crea usando la función <i>array()</i>
Puede contener elementos de distintos tipos	No puede contener elementos de distinto tipo
El anidamiento de listas o estructuras de distinta dimension es posible	Debe contener elementos del mismo tamaño
No se pueden aplicar operaciones aritméticas	Se pueden aplicar operaciones aritméticas directamente
Consume más memoria	Consume comparativamente menos memoria
Mayor flexibilidad a la hora de modificar datos	Menor flexibilidad a la hora de modificar datos



ARRAYS

¿Qué es un Array? Son contenedores que son capaces de guardar más de un objeto al mismo tiempo. Colección ordenada de elementos/objetos.

Lista	Array
Es una estructura in-built	Es una estructura que es necesario importar usando <i>array</i> o <i>numpy</i>
Se crea usando corchetes []	Se crea usando la función <i>array()</i>
Puede contener elementos de distintos tipos	No puede contener elementos de distinto tipo
El anidamiento de estructuras de distinta dimension es posible	Debe contener elementos del mismo tamaño
No se pueden aplicar operaciones aritméticas	Se pueden aplicar operaciones aritméticas directamente
Consume más memoria	Consume comparativamente menos memoria
Mayor flexibilidad a la hora de modificar datos	Menor flexibilidad a la hora de modificar datos

ARRAYS

Lista

No se pueden aplicar operaciones aritméticas

```
my_list = [1,2,3,4,5,6,7,8,9]
nueva_lista = my_list * 2
print(nueva_lista)
```

✓ 0.0s

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Array

Se pueden aplicar operaciones aritméticas directamente

ARRAYS

Lista

No se pueden aplicar operaciones aritméticas

```
my_list = [1,2,3,4,5,6,7,8,9]
nueva_lista = my_list * 2
print(nueva_lista)
```

✓ 0.0s

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Array

Se pueden aplicar operaciones aritméticas directamente

```
my_list = [1,2,3,4,5,6,7,8,9]
nueva_lista = []

for i in range(0,len(my_list)):
    nueva_lista.append(my_list[i] * 2)

print(nueva_lista)
```

✓ 0.0s

```
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

ARRAYS

Lista

No se pueden aplicar operaciones aritméticas

```
my_list = [1,2,3,4,5,6,7,8,9]
nueva_lista = my_list * 2
print(nueva_lista)
```

✓ 0.0s

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
my_list = [1,2,3,4,5,6,7,8,9]
nueva_lista = []

for i in range(0, len(my_list)):
    nueva_lista.append(my_list[i] * 2)

print(nueva_lista)
```

✓ 0.0s

```
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Array

Se pueden aplicar operaciones aritméticas directamente

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
nuevo_array = my_array * 2
print(nuevo_array)
```

✓ 0.0s

```
[ 2  4  6  8 10 12 14 16 18]
```

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
nuevo_array = my_array / 2
print(nuevo_array)
```

✓ 0.2s

```
[0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5]
```

ARRAYS

Lista

Es una estructura in-built

```
lista = [1,2,3,4,5,6,7]
print(type(lista))
print(lista)
```

✓ 0.0s

```
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

Array

Necesitamos usar el modulo array o la libreria numpy

ARRAYS

Lista

Es una estructura in-built

```
lista = [1,2,3,4,5,6,7]
print(type(lista))
print(lista)
```

✓ 0.0s

```
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

```
lista = list([1,2,3,4,5,6,7])
print(type(lista))
print(lista)
```

✓ 0.0s

```
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

Array

Necesitamos usar el modulo array o la libreria numpy

ARRAYS

Lista

Es una estructura in-built

```
lista = [1,2,3,4,5,6,7]
print(type(lista))
print(lista)
```

✓ 0.0s

```
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

```
lista = list([1,2,3,4,5,6,7])
print(type(lista))
print(lista)
```

✓ 0.0s

```
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

Array

Necesitamos usar el modulo array o la libreria numpy

```
my_array = array([1,2,3,4,5,6,7,8,9])
```

⊗ 0.2s

```
-----
NameError                                Traceback (most recent call last)
Cell In[1], line 1
----> 1 my_array = array([1,2,3,4,5,6,7,8,9])

NameError: name 'array' is not defined
```

+ Código

ARRAYS

Lista

Es una estructura in-built

```
lista = [1,2,3,4,5,6,7]
print(type(lista))
print(lista)
```

✓ 0.0s

```
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

```
lista = list([1,2,3,4,5,6,7])
print(type(lista))
print(lista)
```

✓ 0.0s

```
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

Array

Necesitamos usar el modulo array o la libreria numpy

```
my_array = array([1,2,3,4,5,6,7,8,9])
```

⊗ 0.2s

```
-----
NameError                                Traceback (most recent call last)
Cell In[1], line 1
----> 1 my_array = array([1,2,3,4,5,6,7,8,9])

NameError: name 'array' is not defined
```

+ Código

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
```

✓ 0.0s

```
<class 'array.array'>
```

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
print(type(my_array))
```

✓ 0.3s

```
<class 'numpy.ndarray'>
```


ARRAYS

Lista

Se crea usando corchetes

```
lista = [1,2,3,4,5,6,7]
print(type(lista))
print(lista)
```

✓ 0.0s

```
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

```
lista = list([1,2,3,4,5,6,7])
print(type(lista))
print(lista)
```

✓ 0.0s

```
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

Array

Se crea usando la función `array()` (sea del modulo `array` o de la librería `numpy`)

```
my_array = array([1,2,3,4,5,6,7,8,9])
```

⊗ 0.2s

```
-----
NameError                                Traceback (most recent call last)
Cell In[1], line 1
----> 1 my_array = array([1,2,3,4,5,6,7,8,9])

NameError: name 'array' is not defined
```

+ Código

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
```

✓ 0.0s

```
<class 'array.array'>
```

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
print(type(my_array))
```

✓ 0.3s

```
<class 'numpy.ndarray'>
```

ARRAYS

Lista

Puede contener elementos de distinto tipo

Array

No puede contener elementos de distinto tipo

```
lista = ['string', 3, 7.8, True]
print(type(lista))
print(type(lista[0]), type(lista[1]))
```

✓ 0.0s

```
<class 'list'>
<class 'str'> <class 'int'>
```



ARRAYS

Lista

Puede contener elementos de distinto tipo

```
lista = ['string', 3, 7.8, True]
print(type(lista))
print(type(lista[0]), type(lista[1]))
```

✓ 0.0s

```
<class 'list'>
<class 'str'> <class 'int'>
```

Array

No puede contener elementos de distinto tipo

```
import array
my_array = array.array('i', ['string', 3, 7.8, True])
print(type(my_array[0]))
```

⊗ 0.0s

```
-----
TypeError                                Traceback (most recent call last)
Cell In[28], line 2
      1 import array
----> 2 my_array = array.array('i', ['string', 3, 7.8, True])
      3 print(type(my_array[0]))

TypeError: an integer is required (got type str)
```

```
import numpy
my_array = numpy.array(['string', 3, 7.8, True])
print(my_array)
print(type(my_array[2]))
```

✓ 0.0s

```
['string' '3' '7.8' 'True']
<class 'numpy.str_'>
```

ARRAYS

Lista

Anidamiento de listas

```
lista_de_listas = [ ["manzana", "pera", "cereza"], ["string"], ["bmw", "mercedes"] ]
print(type(lista_de_listas))
```

✓ 0.0s

<class 'list'>

```
lista_de_listas = [ ["manzana", "pera", "cereza"],
                    ["string"],
                    ["bmw", "mercedes"] ]

print(type(lista_de_listas))
```

✓ 0.0s

<class 'list'>

Array

Debe contener elementos del mismo tamaño

```
import numpy
my_array = numpy.array(["manzana", "pera", "cereza"],
                       ["string"],
                       ["bmw", "mercedes"])
print(my_array)
print(type(my_array))
```

⊗ 0.0s

TypeError Traceback (most recent call last)

Cell In[39], line 2

```
1 import numpy
----> 2 my_array = numpy.array(["manzana", "pera", "cereza",
3                             ["string"],
4                             ["bmw", "mercedes"]])
5 print(my_array)
6 print(type(my_array))
```

TypeError: array() takes from 1 to 2 positional arguments but 3 were given

¿CUANDO USAMOS UN ARRAY Y CUANDO UNA LISTA?

Lista

- ✓ Nos permite ordenar elementos
 - ✓ Es una estructura fácilmente mutable y flexible
 - ✓ No necesitamos importar ningún módulo
 - Necesita más memoria
 - No se pueden realizar operaciones aritméticas
-
- ➡ Guardar una secuencia pequeña de elementos
 - ➡ No queramos realizar operaciones matemáticas

Array

- ✓ Necesita menos memoria
 - ✓ Se pueden realizar operaciones aritméticas fácilmente
 - Los elementos de un array deben ser del mismo tipo
 - Es una estructura algo menos flexible
-
- ➡ Guardar una secuencia grandes de elementos
 - ➡ Queramos realizar operaciones matemáticas con los datos guardados

MODULO ARRAY

Viene por defecto al instalar Python.

Es un modulo básico para tratar con Arrays.

Sintaxis:

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
```

✓ 0.0s

```
<class 'array.array'>
```

importar modulo *array* llamándolo *arr*

MODULO ARRAY

Viene por defecto al instalar Python.

Es un modulo básico para tratar con Arrays.

Sintaxis:

```
import array as arr  
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])  
print(type(my_array))
```

✓ 0.0s

<class 'array.array'>

importar modulo *array* llamándolo *arr*

del módulo *array* (al que llamamos *arr* a partir de ahora) usamos la función *array()*

MODULO ARRAY

Viene por defecto al instalar Python.

Es un modulo básico para tratar con Arrays.

Sintaxis:

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
```

✓ 0.0s

<class 'array.array'>

importar modulo *array* llamándolo *arr*

del módulo *array* (al que llamamos *arr* a partir de ahora) usamos la función *array()*

indicamos que tipo de objeto va a contener el array. Es este caso enteros simples.

MODULO ARRAY

Viene por defecto al instalar Python.

Es un modulo básico para tratar con Arrays.

Sintaxis:

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
```

✓ 0.0s

<class 'array.array'>

importar modulo *array* llamándolo *arr*

del módulo *array* (al que llamamos *arr* a partir de ahora) usamos la función *array()*

indicamos que tipo de objeto va a contener el array. Es este caso enteros simples.

Contenido del array



MODULO ARRAY

Viene por defecto al instalar Python.

Es un modulo básico para tratar con Arrays.

Sintaxis:

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
```

✓ 0.0s

<class 'array.array'>

Este modulo define un tipo de objeto que representa un arreglo de valores básicos: caracteres, números enteros y de punto flotante. Los arreglos son tipos de secuencias que se comportan de forma similar a las listas, a excepción que el tipo de objeto guardado es definido. El tipo es especificado al momento de crear el objeto mediante *type code*, que es un carácter simple. Se definen los siguientes tipos:

Código de tipo	Tipo C	Tipo Python	Tamaño mínimo en bytes	Notas
'b'	signed char	int	1	
'B'	unsigned char	int	1	
'u'	wchar_t	Carácter unicode	2	(1)
'h'	signed short	int	2	
'H'	unsigned short	int	2	
'i'	signed int	int	2	
'I'	unsigned int	int	2	
'l'	signed long	int	4	
'L'	unsigned long	int	4	
'q'	signed long long	int	8	
'Q'	unsigned long long	int	8	
'f'	float	float	4	
'd'	double	float	8	

<https://docs.python.org/es/3/library/array.html>

signed (con signo) = tipo entero de 32 bits que contiene un numero en el rango [-2147483648, 2147483648].

unsigned (sin signo) = tipo entero de 32 bits que contiene un numero en el rango [0, 4294967295].

LIBRERIA NUMPY

No viene pre-instalada

Es una librería mucho más potente para el tratamiento de arrays.

Sintaxis:

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
print(type(my_array))
```

✓ 0.3s

```
<class 'numpy.ndarray'>
```

importar modulo *numpy* llamándolo *np*

LIBRERIA NUMPY

No viene pre-instalada

Es una librería mucho más potente para el tratamiento de arrays.

Sintaxis:

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
print(type(my_array))
```

✓ 0.3s

```
<class 'numpy.ndarray'>
```

importar modulo *numpy* llamándolo *np*
del módulo *numpy* (al que llamamos *np* a partir de ahora) usamos la función *array()*

LIBRERIA NUMPY

No viene pre-instalada

Es una librería mucho más potente para el tratamiento de arrays.

Sintaxis:

```
import numpy as np  
my_array = np.array([1,2,3,4,5,6,7,8,9])  
print(type(my_array))
```

✓ 0.3s

```
<class 'numpy.ndarray'>
```

importar modulo *numpy* llamándolo *np*

del módulo *numpy* (al que llamamos *np* a partir de ahora) usamos la función *array()*

no necesitamos explicitar el tipo de dato como en el caso del modulo in-built array

INSTALAR NUMPY

Instalación...

Activamos nuestro environment de trabajo:

```
(base) MacBook-Pro-4:Prueba Elena$ conda activate cblocks  
(cblocks) MacBook-Pro-4:Prueba Elena$
```

Dentro del environment de trabajo instalamos numpy:

Podemos usar conda...

```
(cblocks) MacBook-Pro-4:Prueba Elena$ conda install numpy
```

O también pip...

```
(cblocks) MacBook-Pro-4:Prueba Elena$ pip install numpy
```



REPASO

- 1) Qué es un módulo
- 2) Qué es un package/paquete
- 3) Qué es una library/librería
- 4) Qué es un Array y las diferencias con una Lista
- 5) Cómo importar los módulos y librerías relacionadas con Arrays y su sintaxis

CÔNQUER BLOCKS