



Python 3

7. Módulos

Carolina Mañoso, Ángel P. de Madrid y Miguel Romero

Índice

◆ Módulos

◆ Paquetes

◆ Ayuda

◆ Ejemplos de módulos de la biblioteca estándar



Módulos (1/7)

- ◆ Los módulos son entidades que permiten la organización y división lógica de nuestro código.
 - Se corresponde físicamente con los ficheros. Cada archivo de Python almacenado en disco equivale a un módulo.
 - Para tener acceso a las variables y funciones de un módulo hay que importarlo para ello se utiliza la palabra clave `import` seguida del nombre del módulo (nombre el fichero sin extensión).
 - Python tiene una biblioteca de módulos estándar:

<https://docs.python.org/3.8/library/index.html>

O el índice:

<https://docs.python.org/3.8/py-modindex.html>

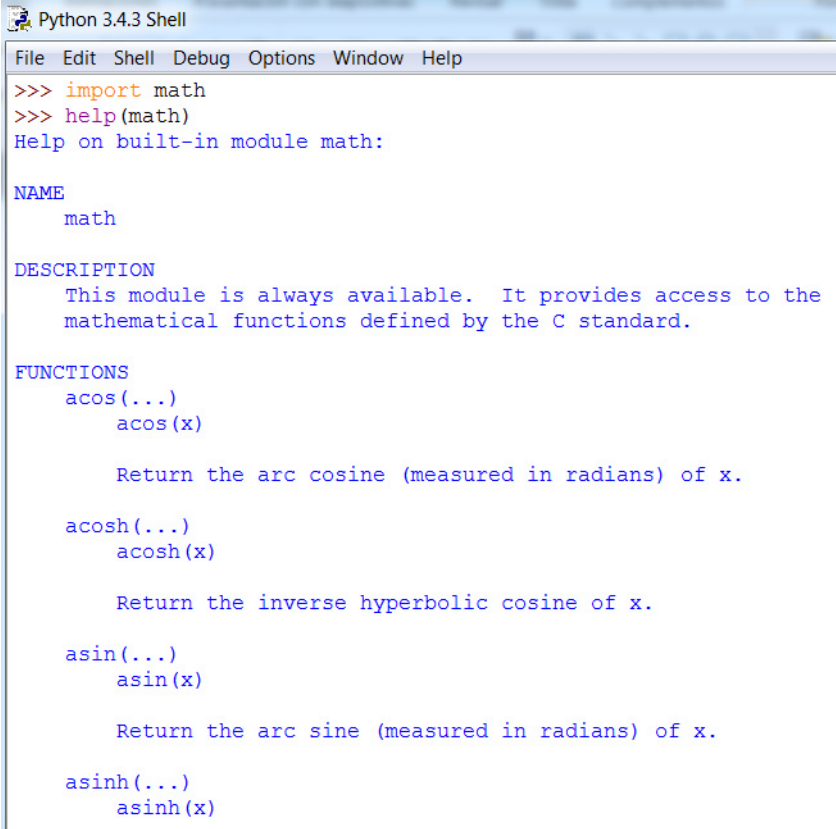
Módulos (2/7)

- Por ejemplo, `math` es un módulo que incorporar variables y funciones matemáticas, si queremos usarlas en nuestro programa haremos:

```
>>>import math
```

- Una vez importado podemos ver que contiene con `help`:

```
>>>help(math)
```



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> import math
>>> help(math)
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module is always available. It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(...)
        acosh(x)

        Return the inverse hyperbolic cosine of x.

    asin(...)
        asin(x)

        Return the arc sine (measured in radians) of x.

    asinh(...)
        asinh(x)
```

Módulos (3/7)

- El importar un módulo genera una variable de tipo objeto con el nombre de ese módulo, `type(math)`. Para usar una variable o una función del módulo, debemos referenciarla a través de esta variable (si no daría error):

```
>>>math.pi
```

```
>>>math.sqrt(9)
```

- En vez de importar todo el módulo podemos importar lo que queremos, exactamente:

```
>>> from math import sqrt, pi
```

- Para usar ahora esta función y variable, lo haremos directamente (ya no tenemos la variable `math`, su uso daría error):

```
>>> sqrt(9)
```

- O podemos importar todo el módulo entero, sin incluir su variable:

```
>>> from math import *
```







- ◆ Si hay nombres de funciones repetidas en los módulos que incorporamos prevalece la última incorporada. (Posibles errores !!!)

Módulos: Práctica (4/7)

◆ Ejemplo que calcula el volumen de la esfera ($\frac{4}{3} \times \Pi \times r^3$):

- Usar el módulo `math` para tener la constante Π :
 - ◆ Importar todo el módulo, versión 1.
 - ◆ Importar solo la variable, versión 2.
 - ◆ Importar todo el interior del módulo, versión 3.

Módulos: Práctica (4/7)

```
#Programa que calcula el volumen de la esfera
#Importa todo el módulo, versión 1
import math 
radio = int(input("introduce el radio de la esfera: "))
volumen = 4.0 / 3.0 * math.pi * radio ** 3 
print ("el volumen es:", volumen)
#-----
#Importa solo la variable versión 2
from math import pi 
radio = int(input("introduce el radio de la esfera: "))
volumen = 4.0 / 3.0 * pi * radio ** 3 
print ("el volumen es:", volumen)
#-----
#Importa todo el interior del módulo versión 3
from math import * 
radio = int(input("introduce el radio de la esfera: "))
volumen = 4.0 / 3.0 * pi * radio ** 3 
print ("el volumen es:", volumen)
```



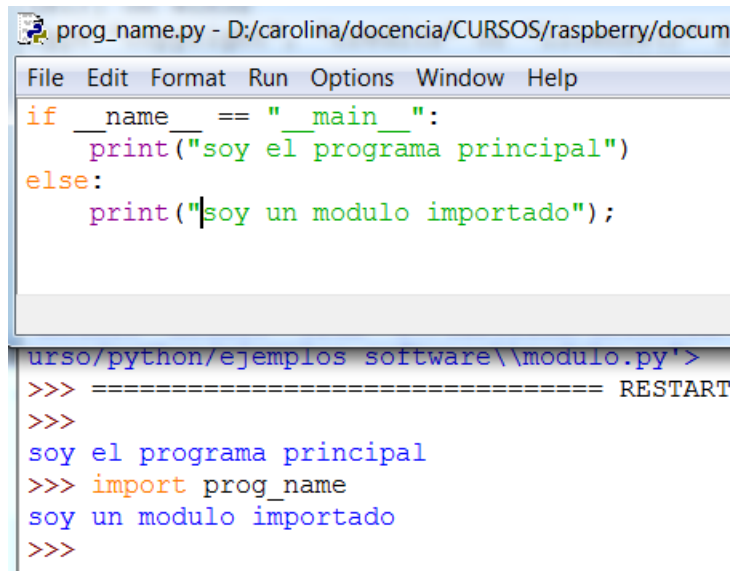
Módulos (5/7)

- Cualquier programa que definamos en un fichero con extensión “.py” puede ser incorporado como módulo en otro.
- Cuando se ejecuta la acción `import modulo`, el módulo se ejecuta, pero sólo la primera vez que se realiza la acción.
- Si se modifica el módulo debe ser cargado otra vez, mediante `restart` la shell o llamando a:

```
>>> imp.reload(modulo)
```


Módulos (6/7)

- Cuando se ejecuta un programa se crea una variable llamada `__name__` cuyo valor es `__main__`.
- Cuando se ejecuta la acción `import modulo`, la variable `__name__` pasa a tener el nombre del módulo.



The screenshot shows a Python IDE window titled 'prog_name.py - D:/carolina/docencia/CURSOS/raspberry/docum'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the editor is:

```
if __name__ == "__main__":
    print("soy el programa principal")
else:
    print("soy un modulo importado");
```

Below the editor, the command prompt shows the execution of the script:

```
urso/python/ejemplos software\\modulo.py'>
>>> ===== RESTART
>>>
soy el programa principal
>>> import prog_name
soy un modulo importado
>>>
```

Módulos (7/7)

- Cuando se importa un módulo, se busca en la lista de directorios contenida en la variable `sys.path`, que son:
 - ◆ El directorio en el que se encuentra el programa que se está ejecutando.
 - ◆ La lista de directorios contenida en la variable `PYTHONPATH`.
 - ◆ El directorio por defecto de la instalación.
- Si no se encuentra ningún módulo con ese nombre se lanza una excepción `ImportError`.

Paquetes (1/2)

- ◆ Los paquetes sirven para organizar los módulos.
 - Son tipos especiales de módulos que permiten agrupar módulos relacionados.
 - Su contrapartida son los directorios.
 - Debe incluir un archivo `__init__.py`.
 - Los paquetes pueden contener subpaquetes.
 - Los módulos no necesariamente deben pertenecer a un paquete.
 - Para cargar un módulo de un paquete:

```
import paquete.modulo # requiere el nombre
```

o

```
from paquete import modulo # no requiere nombre
```

- Los paquetes ya definidos los podemos encontrar en:

<https://pypi.python.org/pypi>

Paquetes (2/2)

PACKAGE INDEX >>

[Browse packages](#)
[List trove classifiers](#)
[RSS \(latest 40 updates\)](#)
[RSS \(newest 40 packages\)](#)
[Terms of Service](#)
[PyPI Tutorial](#)
[PyPI Security](#)
[PyPI Support](#)
[PyPI Bug Reports](#)
[PyPI Discussion](#)
[PyPI Developer Info](#)

[ABOUT >>](#)

[NEWS >>](#)

[DOCUMENTATION >>](#)

[DOWNLOAD >>](#)

[COMMUNITY >>](#)

[FOUNDATION >>](#)

[CORE DEVELOPMENT >>](#)

PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. There are currently **130998** packages here. To contact the PyPI admins, please use the [Support](#) or [Bug reports](#) links.

Get Packages

To use a package from this index either "[pip](#) install *package*" ([get pip](#)) or download, unpack and "[python setup.py install](#)" it.



Infrastructure

To interoperate with the index use the [JSON](#), [XML-RPC](#) or [HTTP](#) interfaces. Use [local mirroring](#) or [caching](#) to make installation more robust.

Package Authors

Submit packages with "[python setup.py upload](#)". You can also use [twine](#)! The index [hosts package docs](#). You must [register](#). Testing? Use [testpypi](#).

Not Logged In

[Login](#)
[Register](#)
[Lost Login?](#)
[Login with OpenID](#) 
[Login with Google](#) 

Status

[Nothing to report](#)

Ayuda (1/4)

- ◆ La función `dir()` lista los nombres (variables, funciones, ...) actualmente definidos:

```
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__']
```

```
>>> import math
```

```
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'math']
```

- Podemos pasarle como argumento cualquier objeto y nos devolverá la lista de los atributos, métodos disponibles, ... para ese objeto:

```
>>> s = 'hola'
```

```
>>> dir(s)
```

Ayuda (2/4)

- Si le pasamos como argumento un módulo, `dir(modulo)`, nos devuelve la lista de nombres definidos en un módulo:

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',
 'hypot', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma',
 'log', 'log10', 'log1p', 'log2', 'modf', 'pi', 'pow',
 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Ayuda (3/4)

◆ La función `help()` nos introduce en `help utility`.

◆ La función `help(object)`, nos proporciona la ayuda sobre ese objeto en particular:

```
>>> help(int)
```

```
>>> s = "hola"
```

```
>>> dir(s)
```

```
>>> help(s.replace)
```

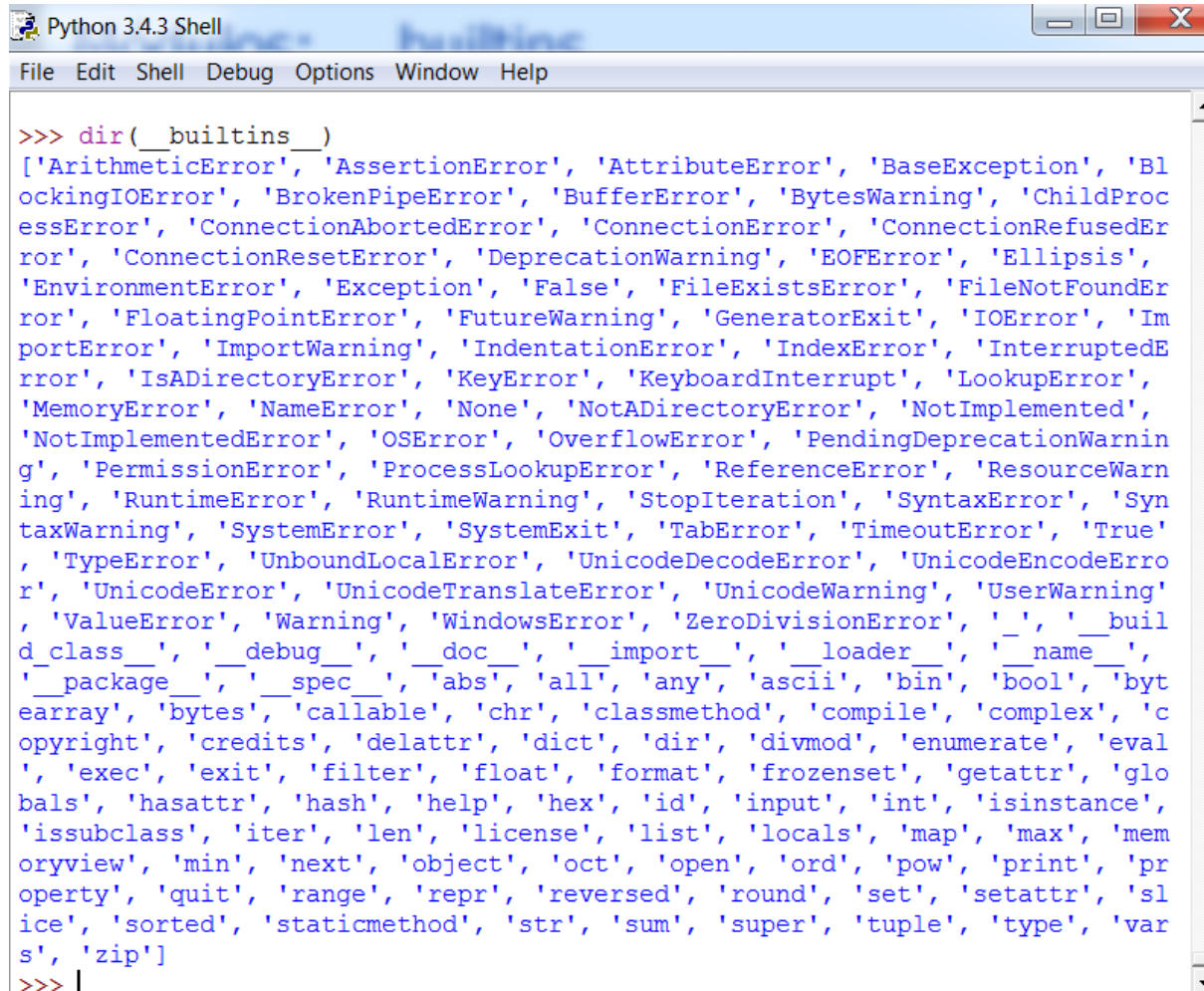
- Si le pasamos como argumento un módulo, `help(modulo)`, nos devuelve toda la información sobre ese módulo:

```
>>> help(math)
```

```
>>> help(math.tan)
```

Ayuda (4/4)

- Las funciones y variables intrínsecas a Python están definidas en el módulo `__builtins__`.



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help

>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '_', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
>>> |
```


Módulos de la biblioteca estándar (1/3)

◆ Para listar los módulos de la biblioteca estándar:
`help("modules")`

◆ Entre otros, encontramos los siguientes módulos:

- El módulo `os` provee funciones para interactuar con el s.o.:

```
>>>import os
```

```
>>>os.getcwd() # devuelve el directorio de trabajo
```

```
os.chdir('/dir1/dir2') # cambia directorio trabajo
```

```
os.chmod(path, mode) # cambia permisos a un archivo
```

```
os.chown(path, uid, gid) # cambia propietario de un archivo
```

```
os.chroot(path) # cambia al directorio de trabajo raíz
```

```
os.getlogin() # devuelve nombre usuario actual
```

```
listado = os.listdir('/home') #lista contenido de directorio
```

```
os.mkdir(path [,mode=511]) # crea subdirectorio
```

Módulos de la biblioteca estándar (2/3)

- El módulo `sys` provee funciones relativas con el propio interprete de Python:

`sys.argv` # devuelve la lista formada por programa y lista de argumentos agregados al ejecutar

`sys.exit()` # fuerza salida del intérprete Python

`sys.getdefaultencoding()` # devuelve codificación de caracteres por defecto

`sys.path` # devuelve paths de Python

`sys.path.append('ruta')` # añade una nueva ruta al path

`sys.modules` # muestra información de los módulos

`sys.version` # obtiene versión de Python

`sys.copyright` # obtiene información de copyright

`sys.platform` # obtiene sistema operativo del sistema

`sys.version_info` # obtiene información de versión

Módulos de la biblioteca estándar (3/3)

- El módulo `time` provee funciones para manipular fechas y horas:

```
time.time() # hora actual en segundos (coma flotante)
```

```
time.ctime() # convierte segundos a cadena
```

```
time.localtime() # muestra hora local como tupla
```

```
time.asctime()) # convierte fecha y hora locales a cadena
```

```
time.sleep() # retarda ejecución un número de segundos
```

- El módulo `math` permite el acceso a las funciones para la matemática de punto flotante.
- El módulo `re` permite procesamiento avanzado de cadenas.
- El módulo `random` provee herramientas para realizar acciones al azar.

Aviso



Python 3 by C. Mañoso, A. P. de Madrid, M. Romero is licensed under a [Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

Esta colección de transparencias se distribuye con fines meramente docentes.

Todas las marcas comerciales y nombres propios de sistemas operativos, programas, hardware, etc. que aparecen en el texto son marcas registradas propiedad de sus respectivas compañías u organizaciones.