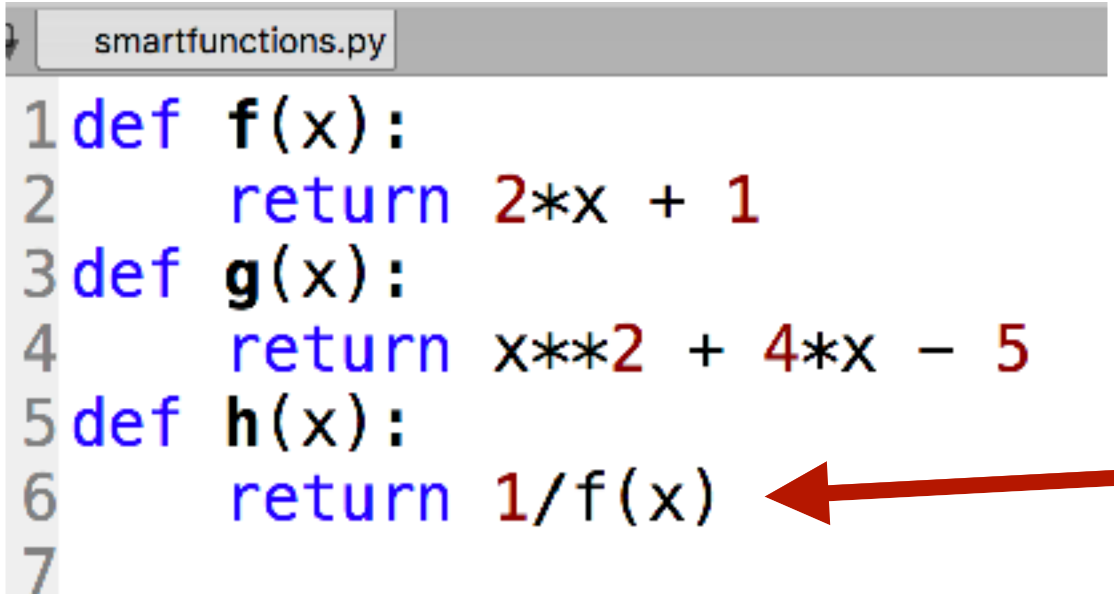
Introduction to Computer Science

Namespaces and Modules

鄒年棣 (Nien-Ti Tsou)

Simple modules – collecting functions

- We can collect functions in a module.
- A Python module is simply a Python source file.
- For example smartfunctions.py.



```python
def f(x):
    return 2*x + 1
def g(x):
    return x**2 + 4*x - 5
def h(x):
    return 1/f(x)

```

But x inside the function is independent.

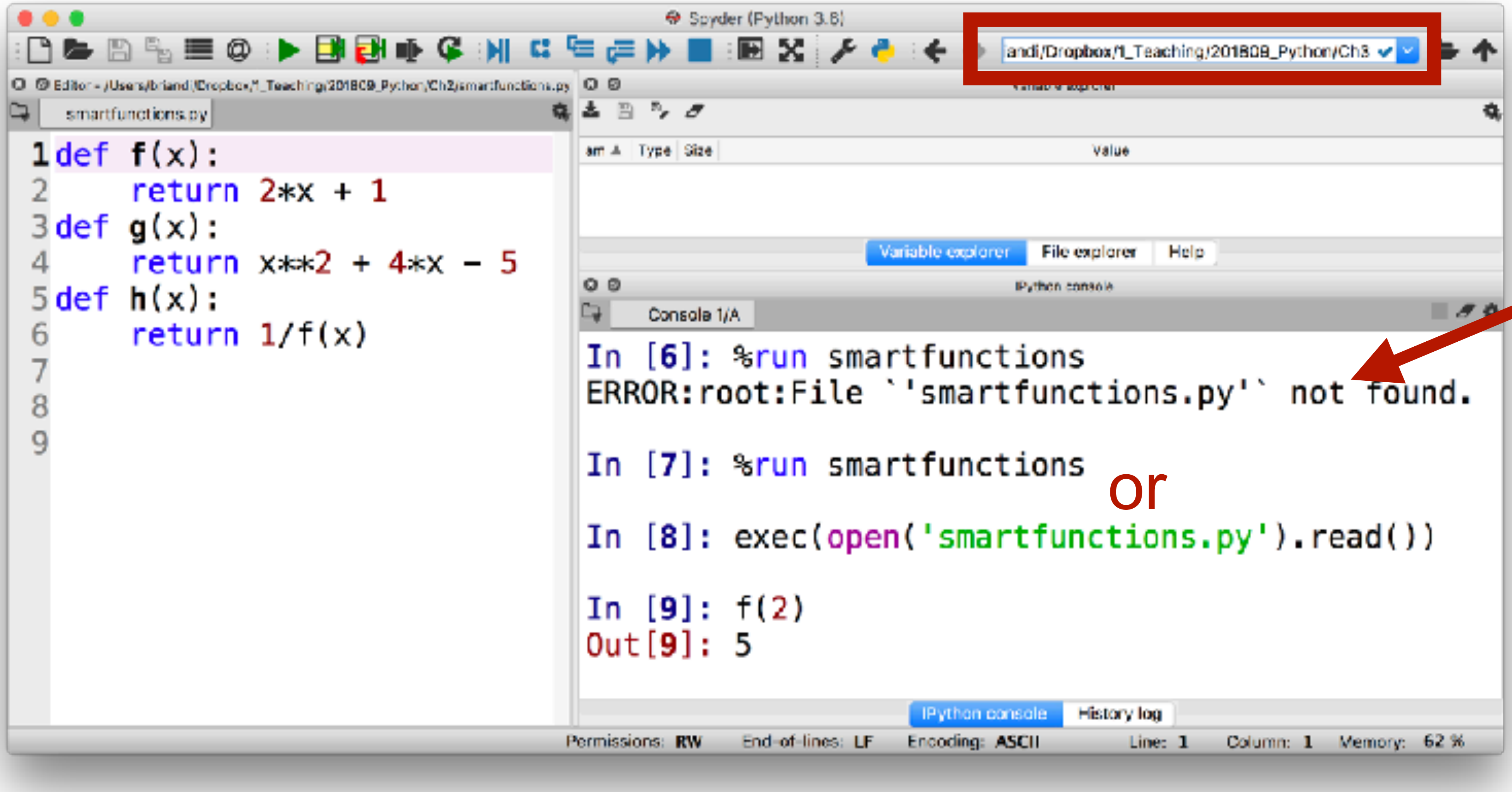
- These functions can be used after it is executed.
- Functions within the module can depend on each other.
- Grouping functions with a common theme or purpose gives modules that can be shared and used by others.

Namespace

- Names of Python objects, such as names of variables, functions, and modules, are collected in namespaces.
- Modules have their own namespaces with the same name as these objects. These namespaces are created when a module is imported. The lifetime of a namespace of a module is as long as the current Python session.
- Functions create its own local namespace when they are executed (invoked). It is deleted when the function stops the execution by a regular return or an exception. Local namespaces are unnamed.

Use your functions and modules

- Change the current folder to where the py-file located.
- Execute it by:
    1. Click play button ▶
    2. `%run smartfunctions`
    3. `exec(open('smartfunctions.py').read())`
- Then you can use the functions in the script or IPython



```python
def f(x):
    return 2*x + 1
def g(x):
    return x**2 + 4*x - 5
def h(x):
    return 1/f(x)
```

```
In [6]: %run smartfunctions
ERROR:root:File `'smartfunctions.py'` not found.

In [7]: %run smartfunctions

In [8]: exec(open('smartfunctions.py').read())

In [9]: f(2)
Out[9]: 5
```

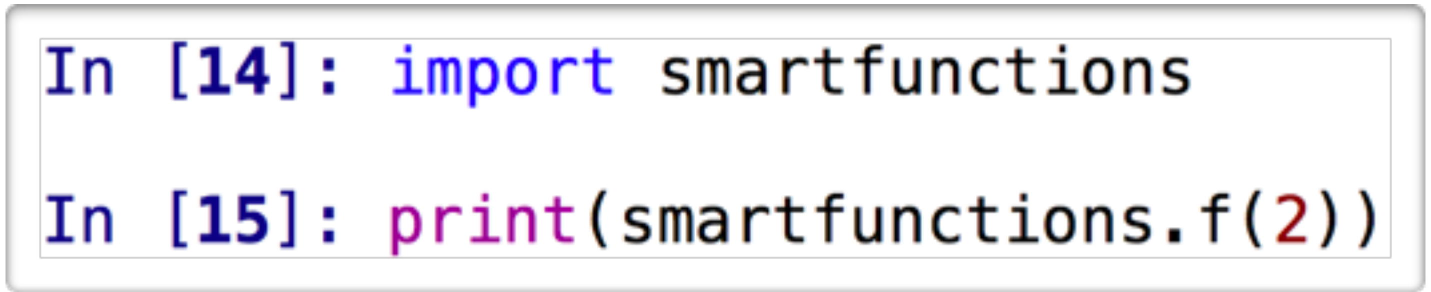
or

Current folder is not set properly

Use your functions and modules

- Alternatively, the modules can be imported by the command `import`. It creates a namespace. The command `from` puts the functions into the general namespace:

- In IPython

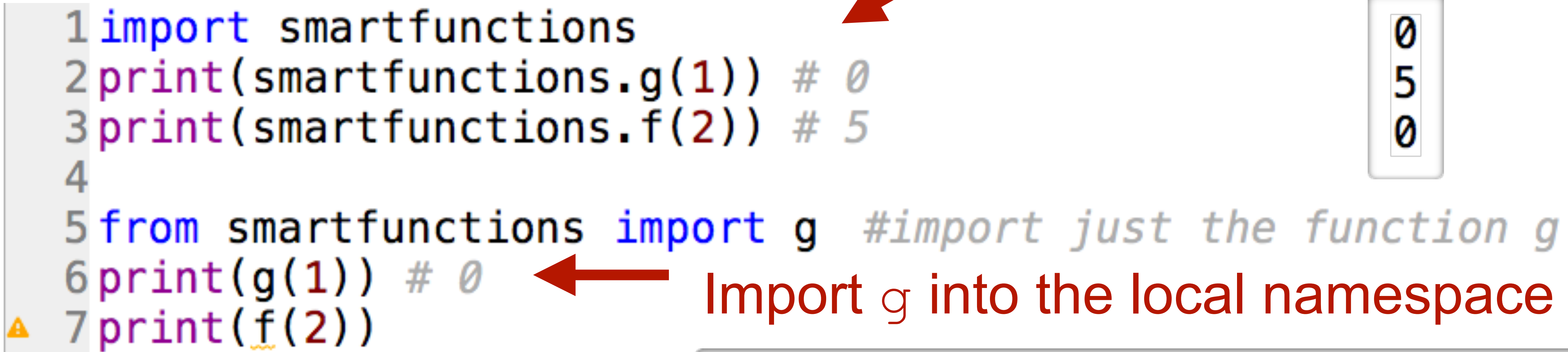


```
In [14]: import smartfunctions

In [15]: print(smartfunctions.f(2))
```

```
5
```

Namespace `smartfunctions` created

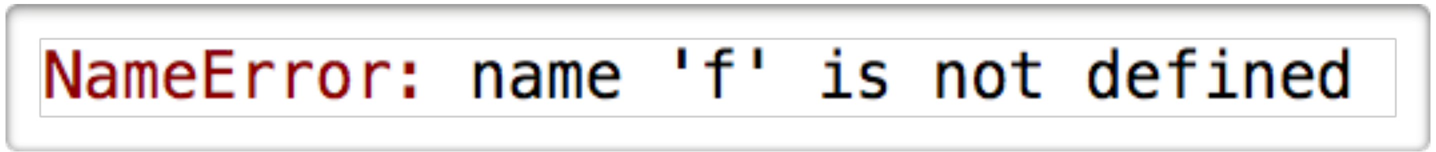
- Or in the script



```
import smartfunctions
print(smartfunctions.g(1)) # 0
print(smartfunctions.f(2)) # 5

from smartfunctions import g #import just the function g
print(g(1)) # 0
print(f(2))
```

```
0
5
0
```

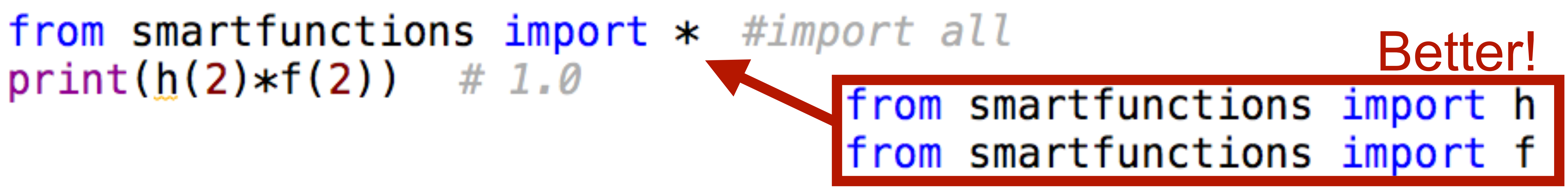
Import g into the local namespace



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

Use your functions and modules efficiently

- Star import `*`: import everything in the modules. It is convenient, but may become difficult for a Python validator to detect undefined names in the program that imported the module. A general best practice, import statements should be as specific as possible and should only import what they need.

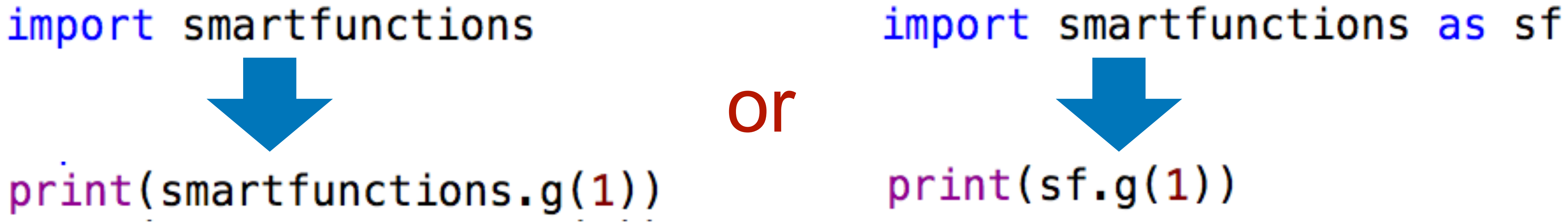


```
from smartfunctions import * #import all
print(h(2)*f(2))   # 1.0
```

Better!

```
from smartfunctions import h
from smartfunctions import f
```

- If too many functions are needed.



```
import smartfunctions
```

```
print(smartfunctions.g(1))
```

or

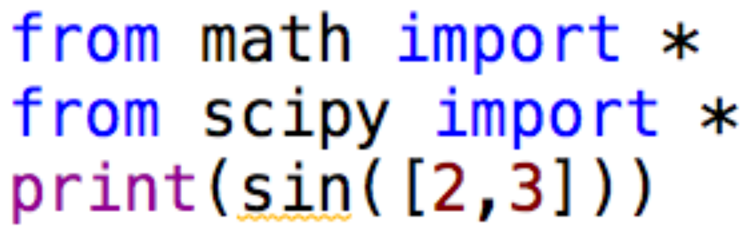
```
import smartfunctions as sf
```

```
print(sf.g(1))
```

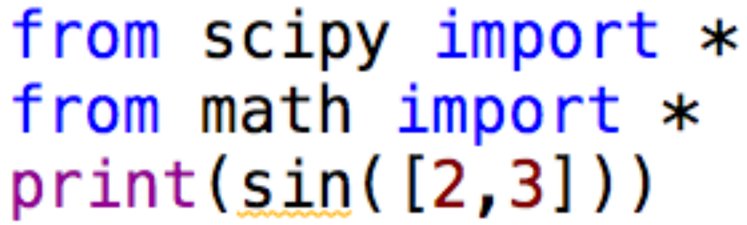
- The commands `import` and `from` import the functions only once into the respective namespace. Changing the functions after the import has no effect for the current Python session.

Example

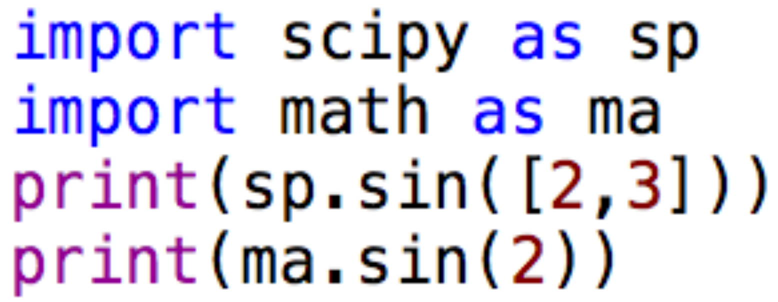
- For example, there are several functions with the name `sin` and they are distinguished by the namespace they belong to.
- They are indeed different, as `scipy.sin` is a universal function accepting numbers, lists or arrays as input, where `math.sin` takes only numbers.



```
from math import *
from scipy import *
print(sin([2,3]))
```

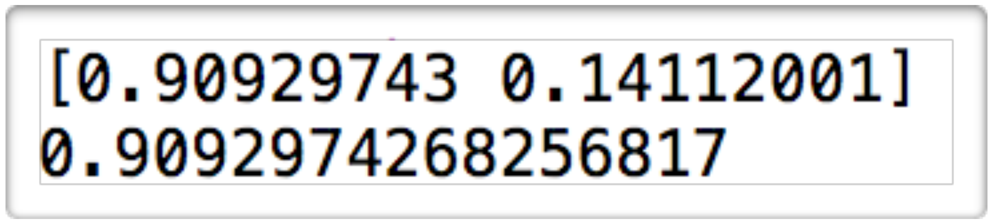


```
from scipy import *
from math import *
print(sin([2,3]))
```



```
import scipy as sp
import math as ma
print(sp.sin([2,3]))
print(ma.sin(2))
```

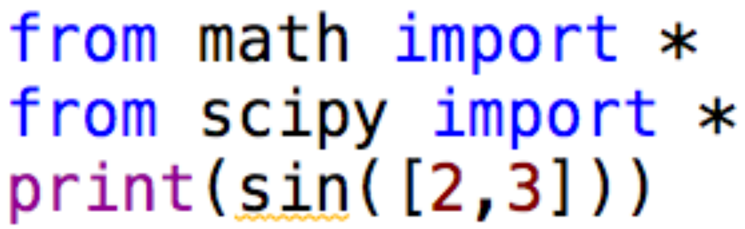
Better!



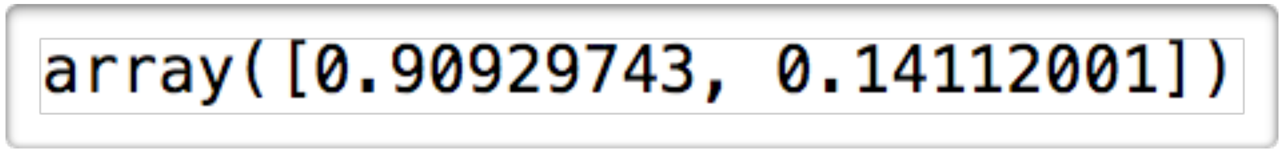
```
[0.90929743 0.14112001]
0.9092974268256817
```

Example

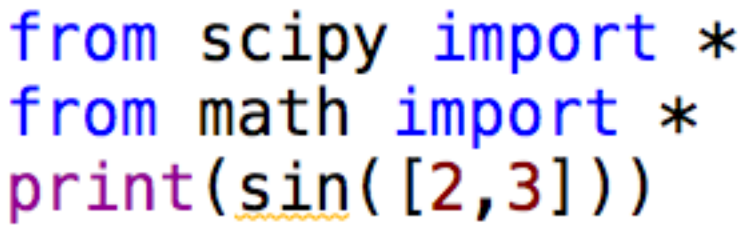
- For example, there are several functions with the name `sin` and they are distinguished by the namespace they belong to.
- They are indeed different, as `scipy.sin` is a universal function accepting numbers, lists or arrays as input, where `math.sin` takes only numbers.



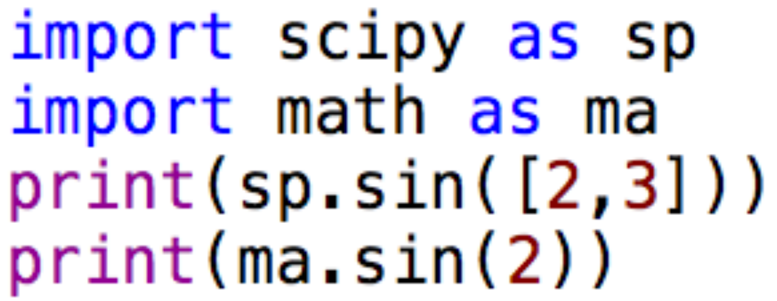
```
from math import *
from scipy import *
print(sin([2,3]))
```



```
array([0.90929743, 0.14112001])
```

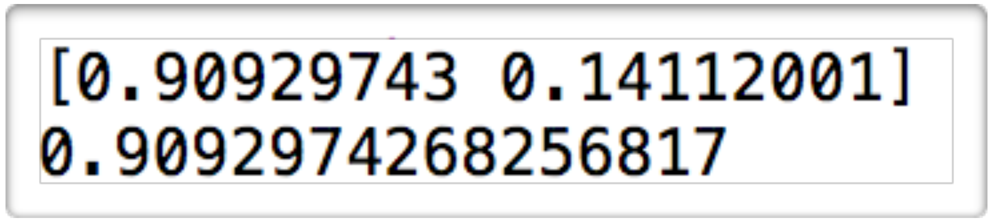


```
from scipy import *
from math import *
print(sin([2,3]))
```



```
import scipy as sp
import math as ma
print(sp.sin([2,3]))
print(ma.sin(2))
```

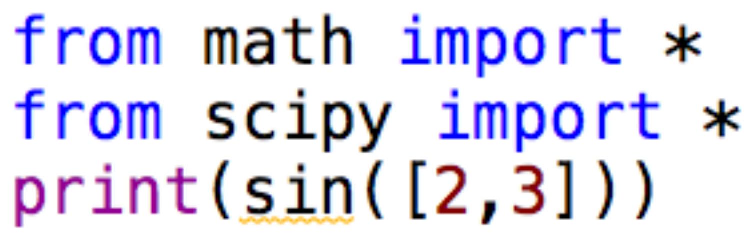
Better!



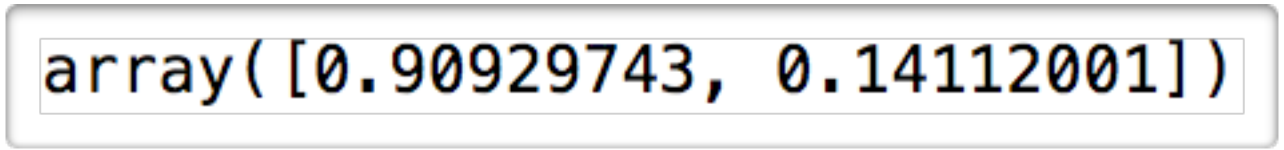
```
[0.90929743 0.14112001]
0.9092974268256817
```

Example

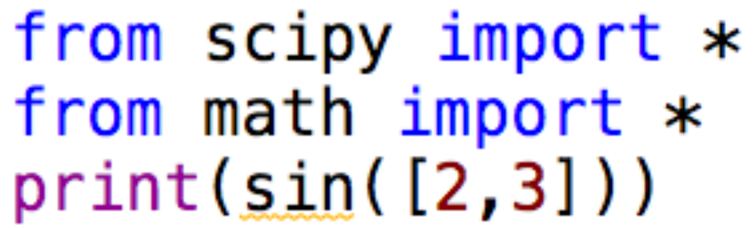
- For example, there are several functions with the name `sin` and they are distinguished by the namespace they belong to.
- They are indeed different, as `scipy.sin` is a universal function accepting numbers, lists or arrays as input, where `math.sin` takes only numbers.



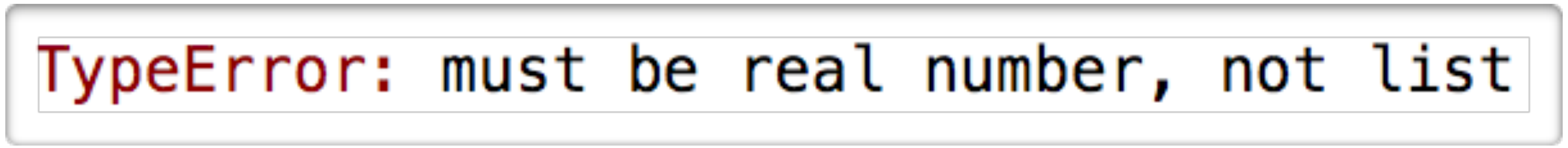
```
from math import *
from scipy import *
print(sin([2,3]))
```



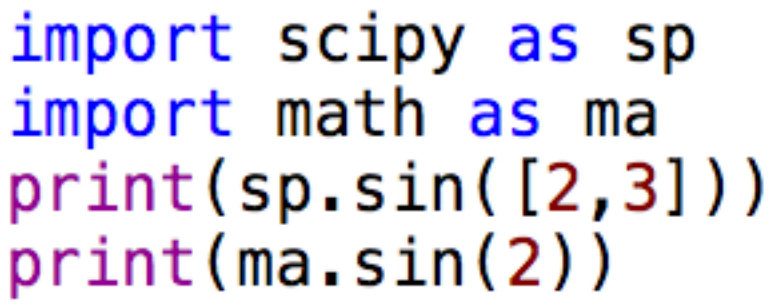
```
array([0.90929743, 0.14112001])
```



```
from scipy import *
from math import *
print(sin([2,3]))
```

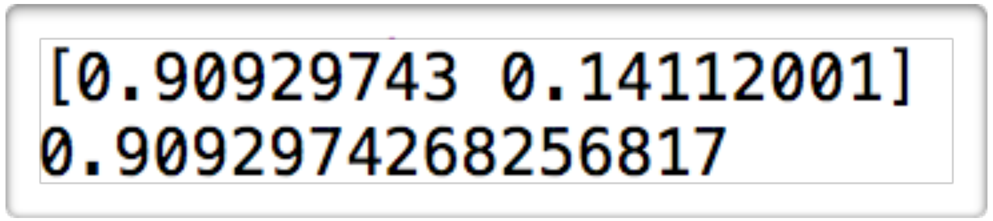


```
TypeError: must be real number, not list
```



```
import scipy as sp
import math as ma
print(sp.sin([2,3]))
print(ma.sin(2))
```

Better!



```
[0.90929743 0.14112001]
0.9092974268256817
```

Example: NumPy, SciPy, and matplotlib

- Python comes with many different libraries by default. You may also want to install more of those for specific purposes, such as optimization, plotting, reading/writing file formats, image handling, and so on. NumPy and SciPy are two important examples of such libraries, matplotlib.

  - Load only certain objects from a library, for example from NumPy:

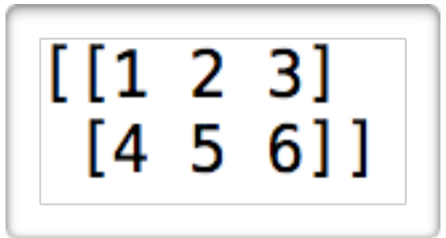
```
from numpy import array, vander
```

  - Or load the entire library:

```
from numpy import *
```

  - Or give access to an entire library by creating a namespace with the library name:

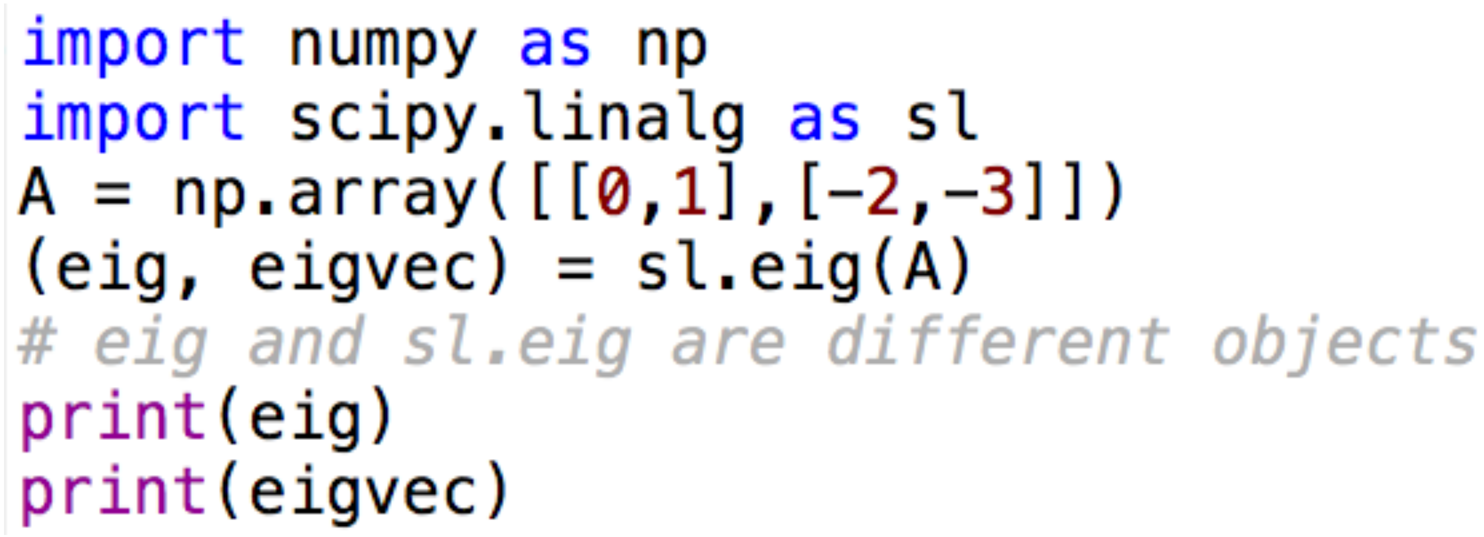
```
import numpy
Matrix = numpy.array([[1, 2, 3],[4, 5, 6]])
print(Matrix)
```



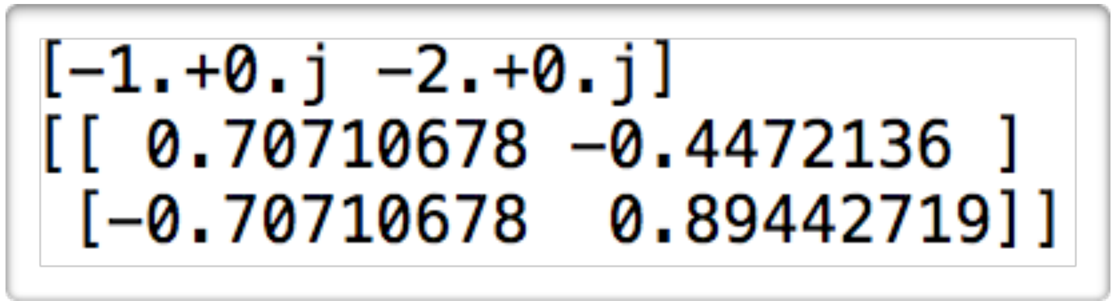
```
[[1 2 3]
 [4 5 6]]
```

Example: NumPy, SciPy, and matplotlib

- `import xxx as xx` affects the readability of your code as well as the possibilities for mistakes.



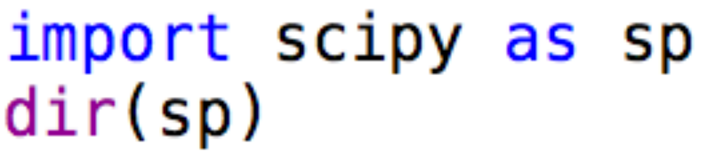
```
import numpy as np
import scipy.linalg as sl
A = np.array([[0,1],[-2,-3]])
(eig, eigvec) = sl.eig(A)
# eig and sl.eig are different objects
print(eig)
print(eigvec)
```



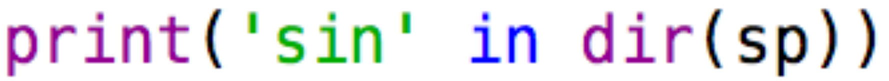
```
[-1.+0.j -2.+0.j]
[[ 0.70710678 -0.4472136 ]
 [-0.70710678  0.89442719]]
```

How many functions in an object?

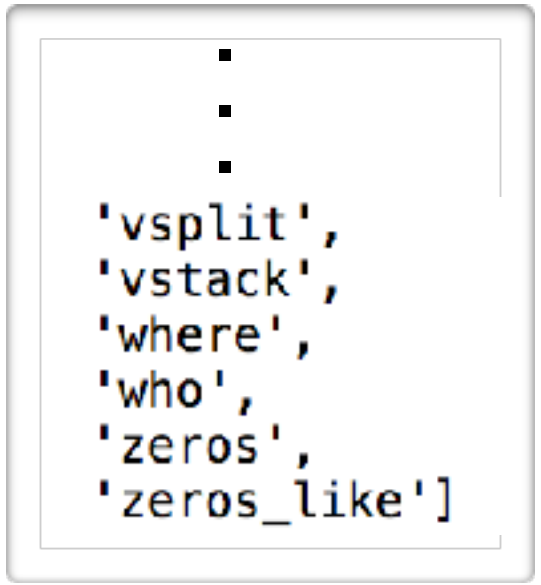
- A list with all the names in a particular namespace can be obtained by the command `dir(…)`.



```
import scipy as sp
dir(sp)
```



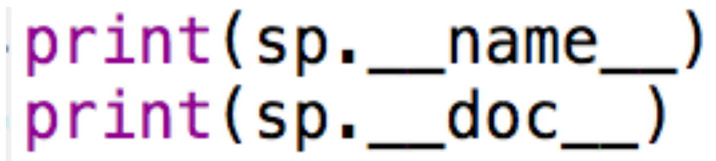
```
print('sin' in dir(sp))
```



```
        .
        .
        .
'vsplit',
'vstack',
'where',
'who',
'zeros',
'zeros_like']
```

Double _

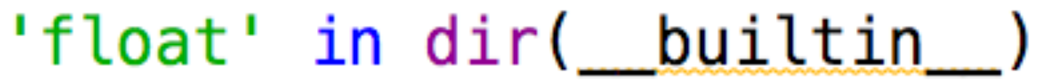
- It contains two special names `__name__` and `__doc__`. The former refers to the name of the module and the latter to its documentation (string).



```
print(sp.__name__)
print(sp.__doc__)
```

Double _

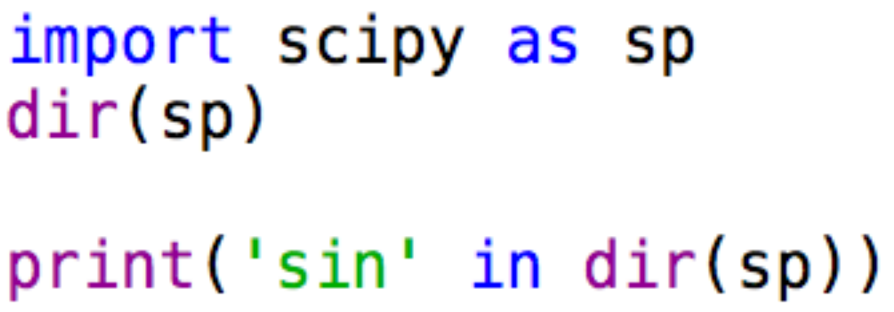
- There is a special namespace, `__builtin__` which contains names that are available in Python without any `import`.



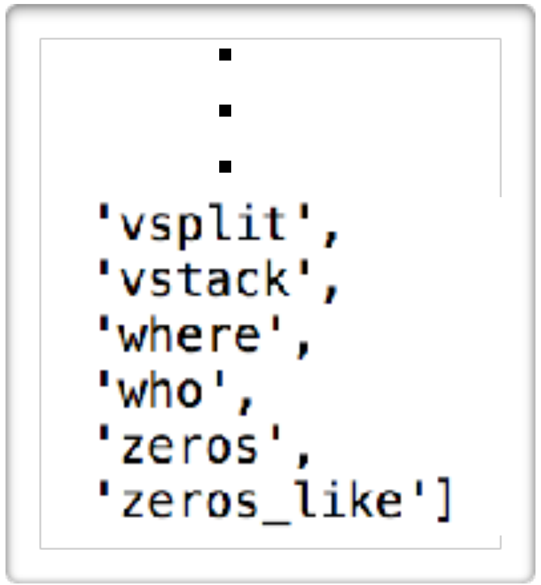
```
'float' in dir(__builtin__)
```

How many functions in an object?

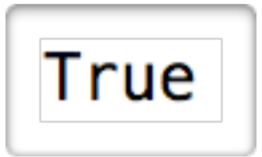
- A list with all the names in a particular namespace can be obtained by the command `dir(…)`.



```
import scipy as sp
dir(sp)

print('sin' in dir(sp))
```



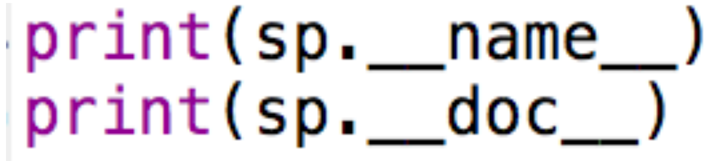
```
    .
    .
    .
'vsplit',
'vstack',
'where',
'who',
'zeros',
'zeros_like']
```



```
True
```

Double _

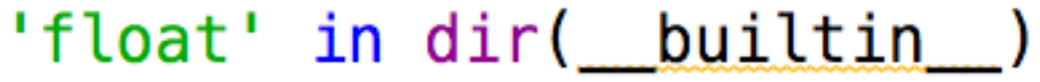
- It contains two special names `__name__` and `__doc__`. The former refers to the name of the module and the latter to its documentation (string).



```
print(sp.__name__)
print(sp.__doc__)
```

Double _

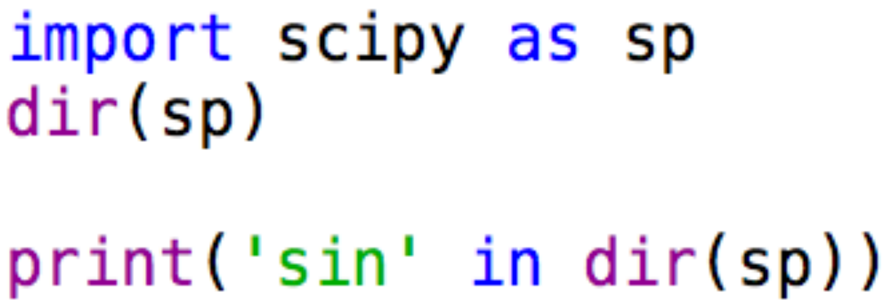
- There is a special namespace, `__builtin__` which contains names that are available in Python without any `import`.



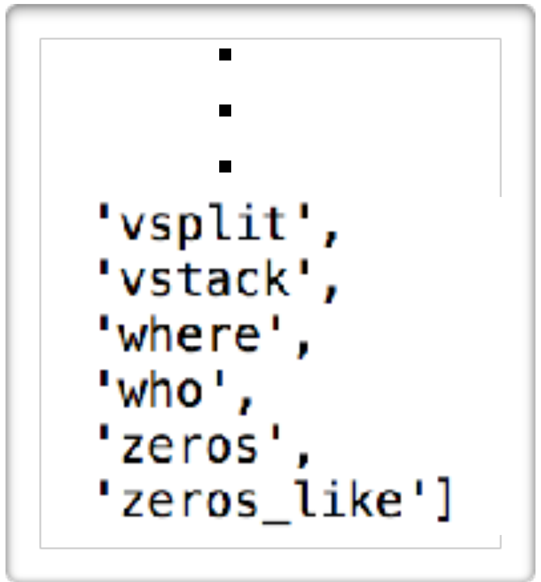
```
'float' in dir(__builtin__)
```

How many functions in an object?

- A list with all the names in a particular namespace can be obtained by the command `dir(…)`.



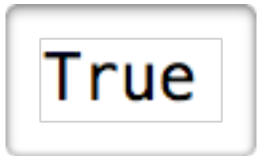
```
import scipy as sp
dir(sp)

print('sin' in dir(sp))
```



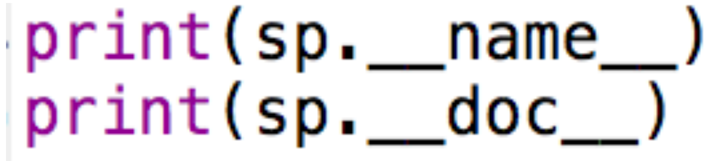
```
    .
    .
    .
 'vsplit',
 'vstack',
 'where',
 'who',
 'zeros',
 'zeros_like']
```

Double _

- It contains two special names `__name__` and `__doc__`. The former refers to the name of the module and the latter to its documentation (string).



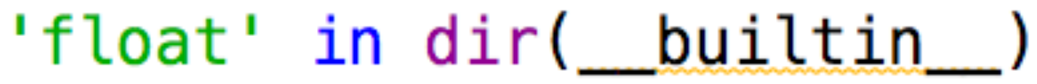
```
True
```



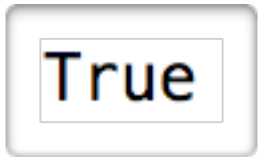
```
print(sp.__name__)
print(sp.__doc__)
```

Double _

- There is a special namespace, `__builtin__` which contains names that are available in Python without any `import`.

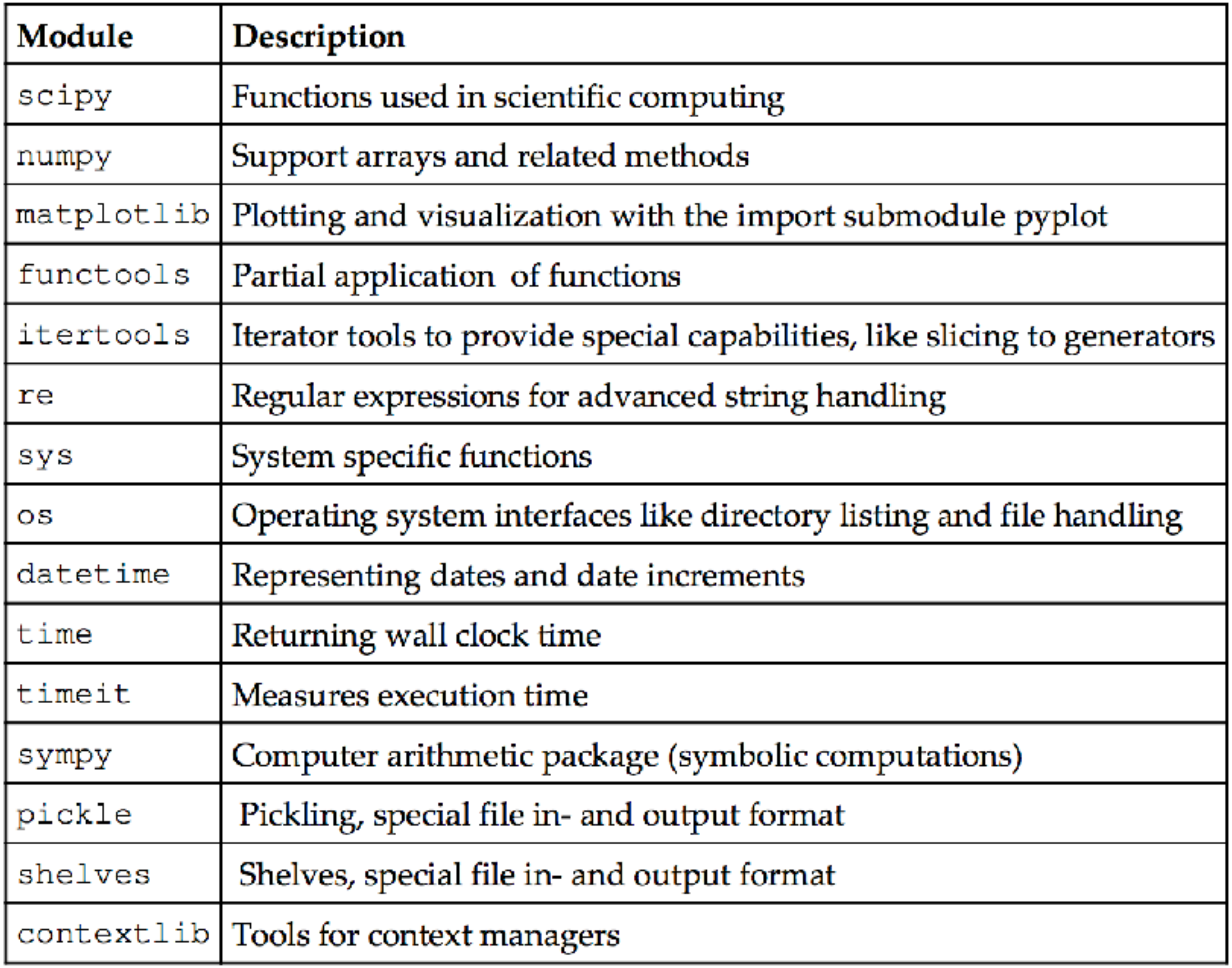


```
'float' in dir(__builtin__)
```



```
True
```

Some useful modules



Module	Description
scipy	Functions used in scientific computing
numpy	Support arrays and related methods
matplotlib	Plotting and visualization with the import submodule pyplot
functools	Partial application of functions
itertools	Iterator tools to provide special capabilities, like slicing to generators
re	Regular expressions for advanced string handling
sys	System specific functions
os	Operating system interfaces like directory listing and file handling
datetime	Representing dates and date increments
time	Returning wall clock time
timeit	Measures execution time
sympy	Computer arithmetic package (symbolic computations)
pickle	Pickling, special file in- and output format
shelves	Shelves, special file in- and output format
contextlib	Tools for context managers