Importing Modules

import Module

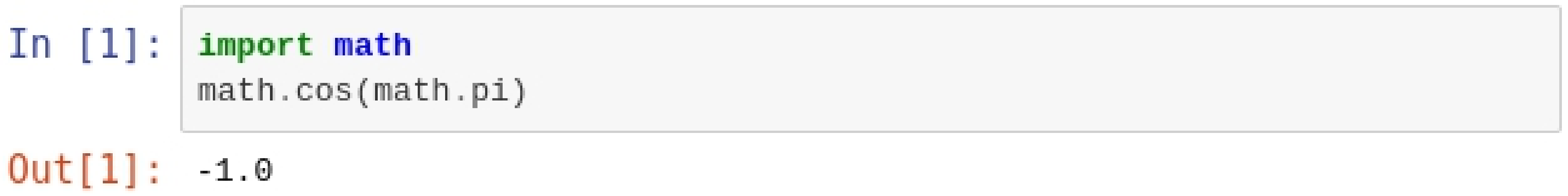
- One feature of Python is that, the Python standard library contains useful tools for a wide range of tasks.
- All Python programs can call a basic set of functions called built-in functions, including the print(), input(), and len() functions you've seen before.
- Python also comes with a set of modules called the standard library.
- Each module is a Python program that contains a related group of functions that can be embedded in your programs.
- For example, the math module has mathematics-related functions, the random module has random number-related functions, and so on.
- Before you can use the functions in a module, you must import the module with an **import** statement.

```
import module1, module2, module3
```

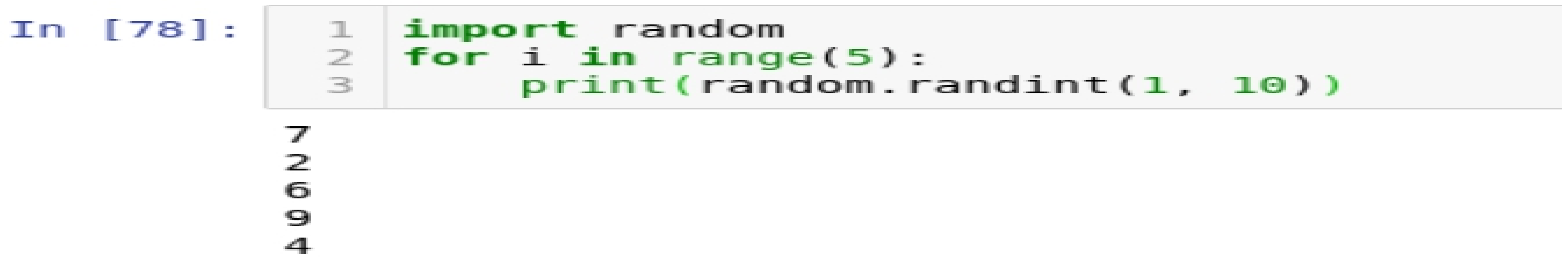
Loading Modules: the import Statement

- For **loading built-in modules**, Python provides the **import** statement.

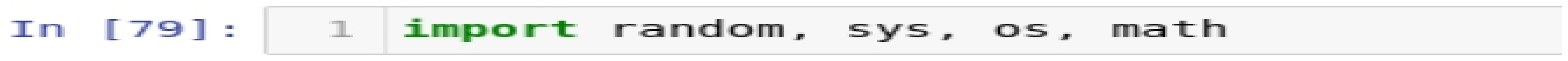
- Explicit import of module contents



```
In [1]:  import math
         math.cos(math.pi)

Out[1]:  -1.0
```



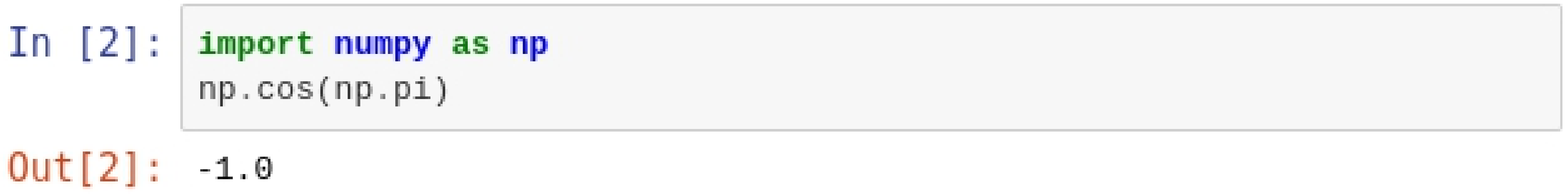
```
In [78]:  import random
          for i in range(5):
              print(random.randint(1, 10))

7
2
6
9
4
```



```
In [79]:  import random, sys, os, math
```

Explicit module import by alias

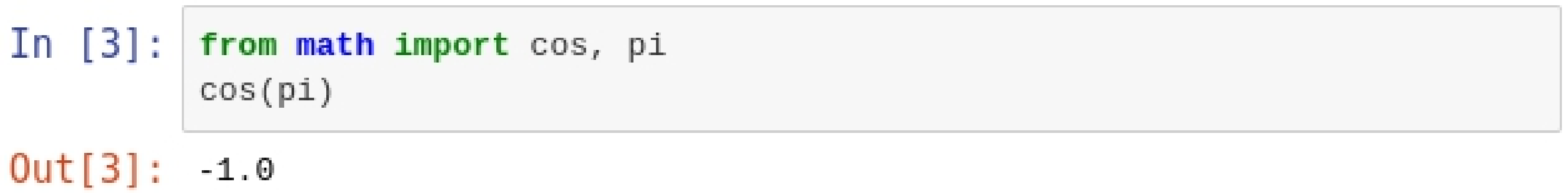
- For longer module names, it's not convenient to use the full module name each time you access some element.
- For this reason, we'll commonly use the "**import ... as ...**" pattern to create a shorter alias for the namespace.
- For example, the **NumPy** (Numerical Python) package is by convention imported under the alias **np**:



```
In [2]: import numpy as np
        np.cos(np.pi)

Out[2]: -1.0
```

Explicit import of module contents

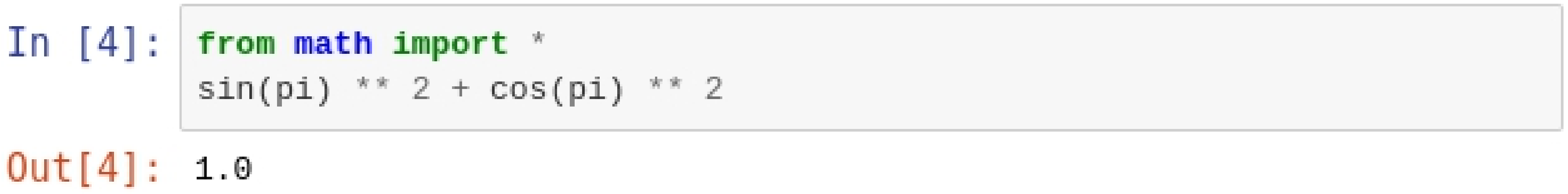
- Sometimes rather than importing the module namespace, you would just like to **import** a few **particular items from the module**.
- This can be done with the "**from ... import ...**" pattern.
- For example, we can import just the cos function and the pi constant from the math module:



```
In [3]: from math import cos, pi
        cos(pi)

Out[3]: -1.0
```

Implicit import of module contents

- It is sometimes useful to **import** the entirety of the **module contents** into the **local namespace**. This can be done with the "**from ... import ***" pattern:

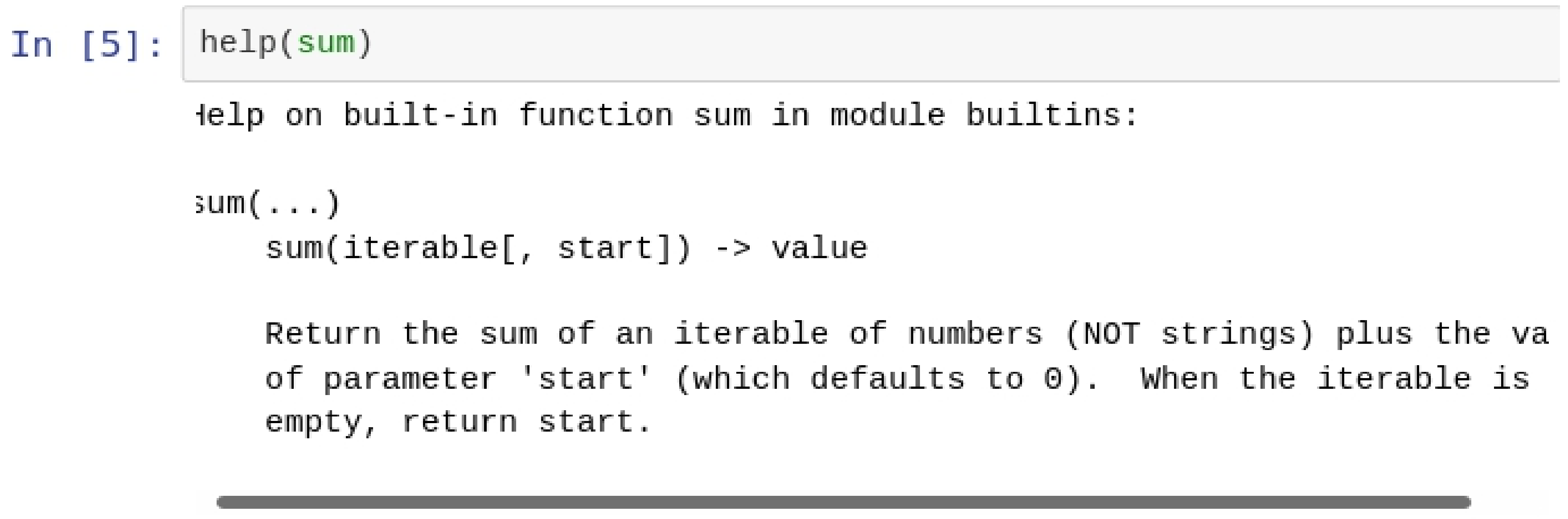


```
In [4]: from math import *
        sin(pi) ** 2 + cos(pi) ** 2

Out[4]: 1.0
```

Problems using *from ... import* *

- The problem is that such imports can **sometimes overwrite function names** that **you do not intend** to **overwrite**, and the implicitness of the statement makes it difficult to determine what has changed.

- For example, Python has a built-in **sum** function that can be used for various operations:



```
In [5]: help(sum)
```

```
Help on built-in function sum in module builtins:

sum(...)
    sum(iterable[, start]) -> value

    Return the sum of an iterable of numbers (NOT strings) plus the va
    of parameter 'start' (which defaults to 0).  When the iterable is
    empty, return start.
```

Problems using *from ... import* *

We can use this to compute the sum of a sequence, starting with a certain value (here, we'll start with `-1`):

```
In [6]: sum(range(5), -1)

Out[6]: 9
```

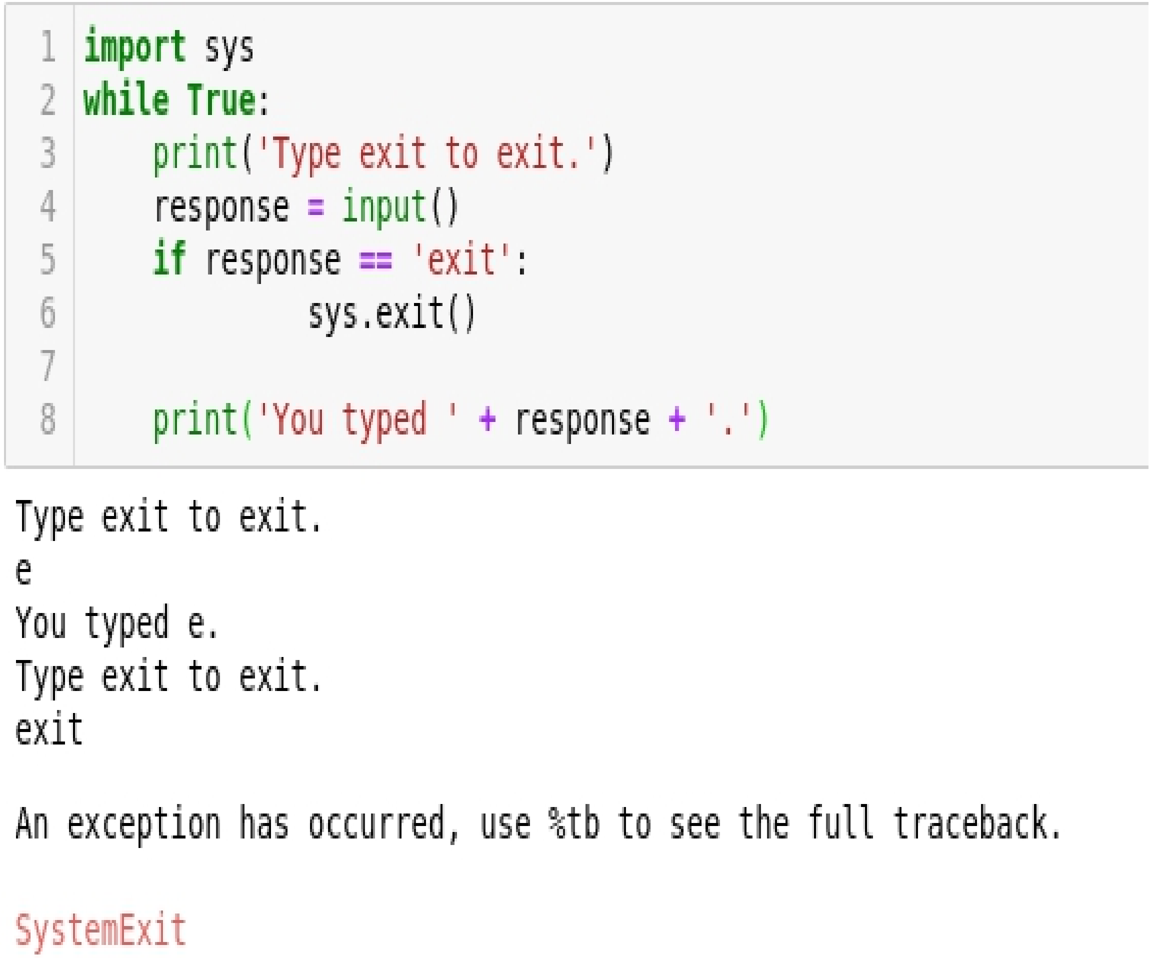
Now observe what happens if we make the *exact same function call* after importing `*` from `numpy`:

```
In [7]: from numpy import *
```

```
In [8]: sum(range(5), -1)

Out[8]: 10
```

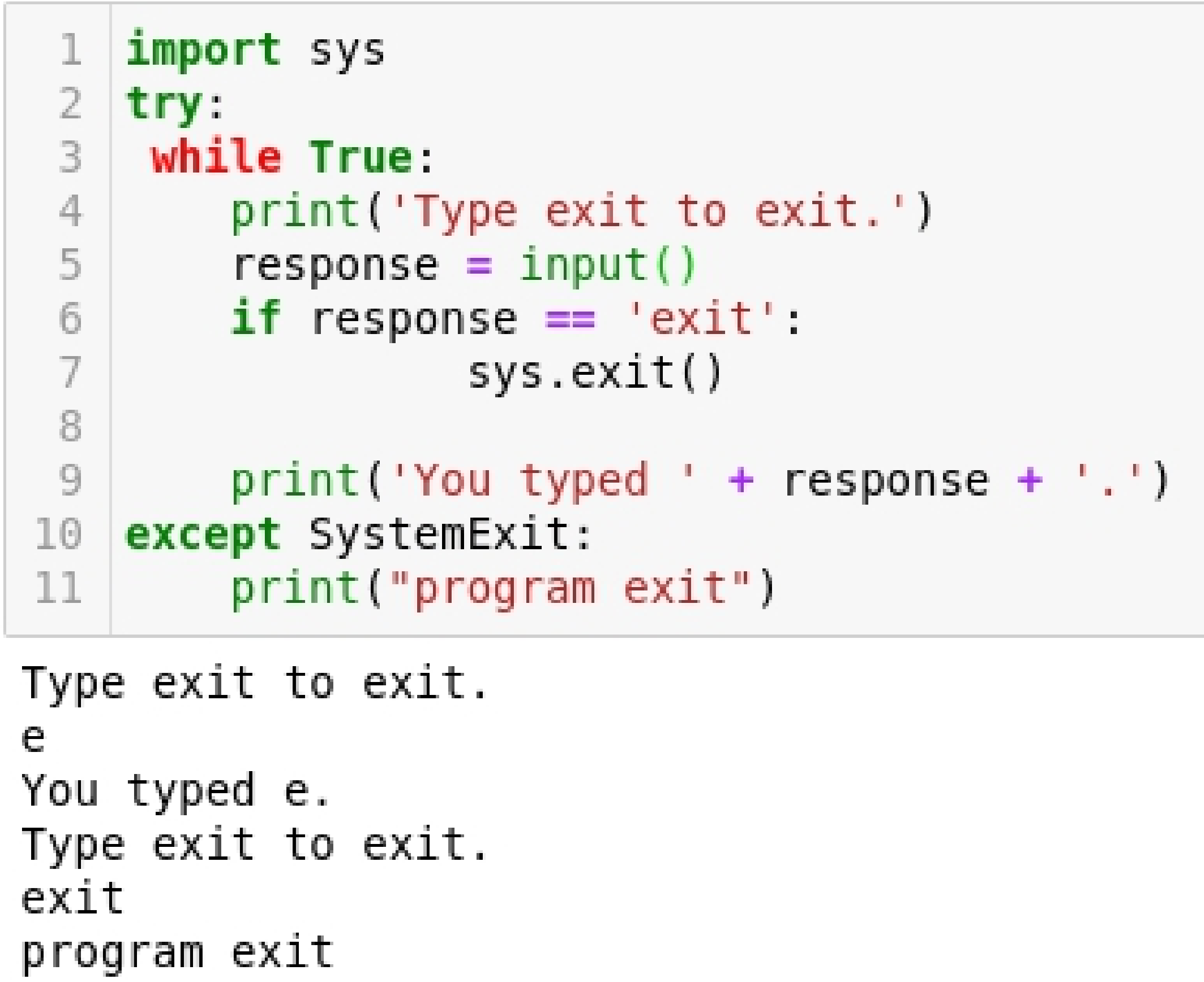
The result is off by one! The reason for this is that the `import *` statement *replaces* the built-in `sum` function with the `numpy.sum` function, which has a different call signature: in the former, we're summing `range(5)` starting at `-1`; in the latter, we're summing `range(5)` along the last axis (indicated by `-1`). This is the type of situation that may arise if care is not taken when using "`import *`" – for this reason, it is best to avoid this unless you know exactly what you are doing.

Example: Ending a Program Early with the sys.exit() Function



```python
import sys
while True:
    print('Type exit to exit.')
    response = input()
    if response == 'exit':
             sys.exit()

    print('You typed ' + response + '.')
```

```
Type exit to exit.
e
You typed e.
Type exit to exit.
exit

An exception has occurred, use %tb to see the full traceback.

SystemExit
```



```python
import sys
try:
 while True:
    print('Type exit to exit.')
    response = input()
    if response == 'exit':
             sys.exit()

    print('You typed ' + response + '.')
except SystemExit:
    print("program exit")
```

```
Type exit to exit.
e
You typed e.
Type exit to exit.
exit
program exit
```

DON'T OVERWRITE MODULE NAMES

When you save your Python scripts, take care not to give them a name that is used by one of Python's modules, such as *random.py*, *sys.py*, *os.py*, or *math.py*. If you accidentally name one of your programs, say, *random.py*, and use an `import random` statement in another program, your program would import your *random.py* file instead of Python's `random` module. This can lead to errors such as `AttributeError: module 'random' has no attribute 'randint'`, since your *random.py* doesn't have the functions that the real `random` module has. Don't use the names of any built-in Python functions either, such as `print()` or `input()`.

Problems like these are uncommon, but can be tricky to solve. As you gain more programming experience, you'll become more aware of the standard names used by Python's modules and functions, and will run into these problems less frequently.

Install Library using *pip*

- Third-party modules or libraries can be installed and managed using Python’s package manager pip.

- The syntax for pip is

pip install module_name