# 6. 模块

如果你从Python解释器退出并再次进入，之前的定义（函数和变量）都会丢失。因此，如果你想编写一个稍长些的程序，最好使用文本编辑器为解释器准备输入并将该文件作为输入运行。这被称作编写 脚本 。随着程序变得越来越长，你或许会想把它拆分成几个文件，以方便维护。你亦或想在不同的程序中使用一个便捷的函数，而不必把这个函数复制到每一个程序中去。

为支持这些，Python有一种方法可以把定义放在一个文件里，并在脚本或解释器的交互式实例中使用它们。这样的文件被称作 模块 ；模块中的定义可以 导入 到其它模块或者 主 模块（你在顶级和计算器模式下执行的脚本中可以访问的变量集合）。

模块是一个包含Python定义和语句的文件。文件名就是模块名后跟文件后缀 .py 。在一个模块内部，模块名（作为一个字符串）可以通过全局变量 \_\_name\_\_ 的值获得。例如，使用你最喜爱的文本编辑器在当前目录下创建一个名为 fibo.py 的文件， 文件中含有以下内容:

*# Fibonacci numbers module*

**def** fib(n): *# write Fibonacci series up to n*

a, b = 0, 1

**while** a < n:

print(a, end=' ')

a, b = b, a+b

print()

**def** fib2(n): *# return Fibonacci series up to n*

result = []

a, b = 0, 1

**while** a < n:

result.append(a)

a, b = b, a+b

**return** result

现在进入Python解释器，并用以下命令导入该模块:

>>>

**>>> import** **fibo**

在当前的符号表中，这并不会直接进入到定义在 fibo 函数内的名称；它只是进入到模块名 fibo 中。你可以用模块名访问这些函数:

>>>

**>>>** fibo.fib(1000)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

**>>>** fibo.fib2(100)

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

**>>>** fibo.\_\_name\_\_

'fibo'

如果你想经常使用某个函数，你可以把它赋值给一个局部变量:

>>>

**>>>** fib = fibo.fib

**>>>** fib(500)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

## 6.1. 有关模块的更多信息

模块可以包含可执行的语句以及函数定义。这些语句用于初始化模块。它们仅在模块 第一次 在 import 语句中被导入时才执行。 [[1]](https://docs.python.org/zh-cn/3/tutorial/modules.html#id2) (当文件被当作脚本运行时，它们也会执行。)

每个模块都有它自己的私有符号表，该表用作模块中定义的所有函数的全局符号表。因此，模块的作者可以在模块内使用全局变量，而不必担心与用户的全局变量发生意外冲突。另一方面，如果你知道自己在做什么，则可以用跟访问模块内的函数的同样标记方法，去访问一个模块的全局变量，modname.itemname。

模块可以导入其它模块。习惯上但不要求把所有 [import](https://docs.python.org/zh-cn/3/reference/simple_stmts.html#import) 语句放在模块（或脚本）的开头。被导入的模块名存放在调入模块的全局符号表中。

[import](https://docs.python.org/zh-cn/3/reference/simple_stmts.html#import) 语句有一个变体，它可以把名字从一个被调模块内直接导入到现模块的符号表里。例如:

>>>

**>>> from** **fibo** **import** fib, fib2

**>>>** fib(500)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

这并不会把被调模块名引入到局部变量表里（因此在这个例子里，fibo 是未被定义的）。

还有一个变体甚至可以导入模块内定义的所有名称:

>>>

**>>> from** **fibo** **import** \*

**>>>** fib(500)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

这会调入所有非以下划线（\_）开头的名称。 在多数情况下，Python程序员都不会使用这个功能，因为它在解释器中引入了一组未知的名称，而它们很可能会覆盖一些你已经定义过的东西。

注意通常情况下从一个模块或者包内调入 \* 的做法是不太被接受的， 因为这通常会导致代码的可读性很差。不过，在交互式编译器中为了节省打字可以这么用。

如果模块名称之后带有 as，则跟在 as 之后的名称将直接绑定到所导入的模块。

>>>

**>>> import** **fibo** **as** **fib**

**>>>** fib.fib(500)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

这会和 import fibo 方式一样有效地调入模块， 唯一的区别是它以 fib 的名称存在的。

It can also be used when utilising [from](https://docs.python.org/zh-cn/3/reference/simple_stmts.html#from) with similar effects:

>>>

**>>> from** **fibo** **import** fib **as** fibonacci

**>>>** fibonacci(500)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

**注解**

出于效率的考虑，每个模块在每个解释器会话中只被导入一次。因此，如果你更改了你的模块，则必须重新启动解释器， 或者，如果它只是一个要交互式地测试的模块，请使用 [importlib.reload()](https://docs.python.org/zh-cn/3/library/importlib.html#importlib.reload)，例如 importimportlib; importlib.reload(modulename)。

### 6.1.1. 以脚本的方式执行模块

当你用下面方式运行一个Python模块:

python fibo.py <arguments>

模块里的代码会被执行，就好像你导入了模块一样，但是 \_\_name\_\_ 被赋值为 "\_\_main\_\_"。 这意味着通过在你的模块末尾添加这些代码:

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** **sys**

fib(int(sys.argv[1]))

你既可以把这个文件当作脚本又可当作一个可调入的模块来使用， 因为那段解析命令行的代码只有在当模块是以“main”文件的方式执行的时候才会运行:

**$** python fibo.py 50

0 1 1 2 3 5 8 13 21 34

如果模块是被导入的，那些代码是不运行的:

>>>

**>>> import** **fibo**

>>>

这经常用于为模块提供一个方便的用户接口，或用于测试（以脚本的方式运行模块从而执行一些测试套件）。

### 6.1.2. 模块搜索路径

当一个名为 spam 的模块被导入的时候，解释器首先寻找具有该名称的内置模块。如果没有找到，然后解释器从 [sys.path](https://docs.python.org/zh-cn/3/library/sys.html#sys.path) 变量给出的目录列表里寻找名为 spam.py 的文件。[sys.path](https://docs.python.org/zh-cn/3/library/sys.html#sys.path) 初始有这些目录地址:

* 包含输入脚本的目录（或者未指定文件时的当前目录）。
* [PYTHONPATH](https://docs.python.org/zh-cn/3/using/cmdline.html#envvar-PYTHONPATH) （一个包含目录名称的列表，它和shell变量 PATH 有一样的语法）。
* 取决于安装的默认设置

**注解**

在支持符号链接的文件系统上，包含输入脚本的目录是在追加符号链接后才计算出来的。换句话说，包含符号链接的目录并 **没有** 被添加到模块的搜索路径上。

在初始化后，Python程序可以更改 [sys.path](https://docs.python.org/zh-cn/3/library/sys.html#sys.path)。包含正在运行脚本的文件目录被放在搜索路径的开头处， 在标准库路径之前。这意味着将加载此目录里的脚本，而不是标准库中的同名模块。 除非有意更换，否则这是错误。更多信息请参阅 [标准模块](https://docs.python.org/zh-cn/3/tutorial/modules.html#tut-standardmodules)。

### 6.1.3. “编译过的”Python文件

为了加速模块载入，Python在 \_\_pycache\_\_ 目录里缓存了每个模块的编译后版本，名称为 module.*version*.pyc ，其中名称中的版本字段对编译文件的格式进行编码； 它一般使用Python版本号。例如，在CPython版本3.3中，spam.py的编译版本将被缓存为 \_\_pycache\_\_/spam.cpython-33.pyc。此命名约定允许来自不同发行版和不同版本的Python的已编译模块共存。

Python根据编译版本检查源的修改日期，以查看它是否已过期并需要重新编译。这是一个完全自动化的过程。此外，编译的模块与平台无关，因此可以在具有不同体系结构的系统之间共享相同的库。

Python在两种情况下不会检查缓存。首先，对于从命令行直接载入的模块，它从来都是重新编译并且不存储编译结果；其次，如果没有源模块，它不会检查缓存。为了支持无源文件（仅编译）发行版本， 编译模块必须是在源目录下，并且绝对不能有源模块。

给专业人士的一些小建议:

* 你可以在Python命令中使用 [-O](https://docs.python.org/zh-cn/3/using/cmdline.html#cmdoption-o) 或者 [-OO](https://docs.python.org/zh-cn/3/using/cmdline.html#cmdoption-oo) 开关， 以减小编译后模块的大小。 -O 开关去除断言语句，-OO 开关同时去除断言语句和 \_\_doc\_\_ 字符串。由于有些程序可能依赖于这些，你应当只在清楚自己在做什么时才使用这个选项。“优化过的”模块有一个 opt- 标签并且通常小些。将来的发行版本或许会更改优化的效果。
* 一个从 .pyc 文件读出的程序并不会比它从 .py 读出时运行的更快，.pyc 文件唯一快的地方在于载入速度。
* [compileall](https://docs.python.org/zh-cn/3/library/compileall.html#module-compileall) 模块可以为一个目录下的所有模块创建.pyc文件。
* 关于这个过程，[**PEP 3147**](https://www.python.org/dev/peps/pep-3147) 中有更多细节，包括一个决策流程图。

## 6.2. 标准模块

Python附带了一个标准模块库，在单独的文档Python库参考（以下称为“库参考”）中进行了描述。一些模块内置于解释器中；它们提供对不属于语言核心但仍然内置的操作的访问，以提高效率或提供对系统调用等操作系统原语的访问。这些模块的集合是一个配置选项，它也取决于底层平台。例如，[winreg](https://docs.python.org/zh-cn/3/library/winreg.html#module-winreg) 模块只在Windows操作系统上提供。一个特别值得注意的模块 [sys](https://docs.python.org/zh-cn/3/library/sys.html#module-sys)，它被内嵌到每一个Python解释器中。变量 sys.ps1 和 sys.ps2 定义用作主要和辅助提示的字符串:

>>>

**>>> import** **sys**

**>>>** sys.ps1

'>>> '

**>>>** sys.ps2

'... '

**>>>** sys.ps1 = 'C> '

C> print('Yuck!')

Yuck!

C>

这两个变量只有在编译器是交互模式下才被定义。

sys.path 变量是一个字符串列表，用于确定解释器的模块搜索路径。该变量被初始化为从环境变量 [PYTHONPATH](https://docs.python.org/zh-cn/3/using/cmdline.html#envvar-PYTHONPATH) 获取的默认路径，或者如果 [PYTHONPATH](https://docs.python.org/zh-cn/3/using/cmdline.html#envvar-PYTHONPATH) 未设置，则从内置默认路径初始化。你可以使用标准列表操作对其进行修改:

>>>

**>>> import** **sys**

**>>>** sys.path.append('/ufs/guido/lib/python')

## 6.3. [dir()](https://docs.python.org/zh-cn/3/library/functions.html#dir) 函数

内置函数 [dir()](https://docs.python.org/zh-cn/3/library/functions.html#dir) 用于查找模块定义的名称。 它返回一个排序过的字符串列表:

>>>

**>>> import** **fibo**, **sys**

**>>>** dir(fibo)

['\_\_name\_\_', 'fib', 'fib2']

**>>>** dir(sys)

['\_\_displayhook\_\_', '\_\_doc\_\_', '\_\_excepthook\_\_', '\_\_loader\_\_', '\_\_name\_\_',

'\_\_package\_\_', '\_\_stderr\_\_', '\_\_stdin\_\_', '\_\_stdout\_\_',

'\_clear\_type\_cache', '\_current\_frames', '\_debugmallocstats', '\_getframe',

'\_home', '\_mercurial', '\_xoptions', 'abiflags', 'api\_version', 'argv',

'base\_exec\_prefix', 'base\_prefix', 'builtin\_module\_names', 'byteorder',

'call\_tracing', 'callstats', 'copyright', 'displayhook',

'dont\_write\_bytecode', 'exc\_info', 'excepthook', 'exec\_prefix',

'executable', 'exit', 'flags', 'float\_info', 'float\_repr\_style',

'getcheckinterval', 'getdefaultencoding', 'getdlopenflags',

'getfilesystemencoding', 'getobjects', 'getprofile', 'getrecursionlimit',

'getrefcount', 'getsizeof', 'getswitchinterval', 'gettotalrefcount',

'gettrace', 'hash\_info', 'hexversion', 'implementation', 'int\_info',

'intern', 'maxsize', 'maxunicode', 'meta\_path', 'modules', 'path',

'path\_hooks', 'path\_importer\_cache', 'platform', 'prefix', 'ps1',

'setcheckinterval', 'setdlopenflags', 'setprofile', 'setrecursionlimit',

'setswitchinterval', 'settrace', 'stderr', 'stdin', 'stdout',

'thread\_info', 'version', 'version\_info', 'warnoptions']

如果没有参数，[dir()](https://docs.python.org/zh-cn/3/library/functions.html#dir) 会列出你当前定义的名称:

>>>

**>>>** a = [1, 2, 3, 4, 5]

**>>> import** **fibo**

**>>>** fib = fibo.fib

**>>>** dir()

['\_\_builtins\_\_', '\_\_name\_\_', 'a', 'fib', 'fibo', 'sys']

注意：它列出所有类型的名称：变量，模块，函数，等等。

[dir()](https://docs.python.org/zh-cn/3/library/functions.html#dir) 不会列出内置函数和变量的名称。如果你想要这些，它们的定义是在标准模块 [builtins](https://docs.python.org/zh-cn/3/library/builtins.html#module-builtins) 中:

>>>

**>>> import** **builtins**

**>>>** 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', 'ZeroDivisionError', '\_', '\_\_build\_class\_\_',

'\_\_debug\_\_', '\_\_doc\_\_', '\_\_import\_\_', '\_\_name\_\_', '\_\_package\_\_', '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']

## 6.4. 包

包是一种通过用“带点号的模块名”来构造 Python 模块命名空间的方法。 例如，模块名 A.B 表示 A 包中名为 B 的子模块。正如模块的使用使得不同模块的作者不必担心彼此的全局变量名称一样，使用加点的模块名可以使得 NumPy 或 Pillow 等多模块软件包的作者不必担心彼此的模块名称一样。

假设你想为声音文件和声音数据的统一处理，设计一个模块集合（一个“包”）。由于存在很多不同的声音文件格式（通常由它们的扩展名来识别，例如：.wav， .aiff， .au），因此为了不同文件格式间的转换，你可能需要创建和维护一个不断增长的模块集合。 你可能还想对声音数据还做很多不同的处理（例如，混声，添加回声，使用均衡器功能，创造人工立体声效果）， 因此为了实现这些处理，你将另外写一个无穷尽的模块流。这是你的包的可能结构（以分层文件系统的形式表示）：

sound/ Top-level package

\_\_init\_\_.py Initialize the sound package

formats/ Subpackage for file format conversions

\_\_init\_\_.py

wavread.py

wavwrite.py

aiffread.py

aiffwrite.py

auread.py

auwrite.py

...

effects/ Subpackage for sound effects

\_\_init\_\_.py

echo.py

surround.py

reverse.py

...

filters/ Subpackage for filters

\_\_init\_\_.py

equalizer.py

vocoder.py

karaoke.py

...

当导入这个包时，Python搜索 sys.path 里的目录，查找包的子目录。

必须要有 \_\_init\_\_.py 文件才能让 Python 将包含该文件的目录当作包。 这样可以防止具有通常名称例如 string的目录在无意中隐藏稍后在模块搜索路径上出现的有效模块。 在最简单的情况下，\_\_init\_\_.py 可以只是一个空文件，但它也可以执行包的初始化代码或设置 \_\_all\_\_ 变量，具体将在后文介绍。

包的用户可以从包中导入单个模块，例如:

**import** **sound.effects.echo**

这会加载子模块 sound.effects.echo 。但引用它时必须使用它的全名。

sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)

导入子模块的另一种方法是

**from** **sound.effects** **import** echo

这也会加载子模块 echo ，并使其在没有包前缀的情况下可用，因此可以按如下方式使用:

echo.echofilter(input, output, delay=0.7, atten=4)

另一种形式是直接导入所需的函数或变量:

**from** **sound.effects.echo** **import** echofilter

同样，这也会加载子模块 echo，但这会使其函数 echofilter() 直接可用:

echofilter(input, output, delay=0.7, atten=4)

请注意，当使用 from package import item 时，item可以是包的子模块（或子包），也可以是包中定义的其他名称，如函数，类或变量。 import 语句首先测试是否在包中定义了item；如果没有，它假定它是一个模块并尝试加载它。如果找不到它，则引发 [ImportError](https://docs.python.org/zh-cn/3/library/exceptions.html#ImportError) 异常。

相反，当使用 import item.subitem.subsubitem 这样的语法时，除了最后一项之外的每一项都必须是一个包；最后一项可以是模块或包，但不能是前一项中定义的类或函数或变量。

### 6.4.1. 从包中导入 \*

当用户写 from sound.effects import \* 会发生什么？理想情况下，人们希望这会以某种方式传递给文件系统，找到包中存在哪些子模块，并将它们全部导入。这可能需要很长时间，导入子模块可能会产生不必要的副作用，这种副作用只有在显式导入子模块时才会发生。

唯一的解决方案是让包作者提供一个包的显式索引。[import](https://docs.python.org/zh-cn/3/reference/simple_stmts.html#import) 语句使用下面的规范：如果一个包的 \_\_init\_\_.py 代码定义了一个名为 \_\_all\_\_ 的列表，它会被视为在遇到 from package import \* 时应该导入的模块名列表。在发布该包的新版本时，包作者可以决定是否让此列表保持更新。包作者如果认为从他们的包中导入 \* 的操作没有必要被使用，也可以决定不支持此列表。例如，文件 sound/effects/\_\_init\_\_.py 可以包含以下代码:

\_\_all\_\_ = ["echo", "surround", "reverse"]

这意味着 from sound.effects import \* 将导入 sound 包的三个命名子模块。

如果没有定义 \_\_all\_\_，from sound.effects import \* 语句 不 会从包 sound.effects 中导入所有子模块到当前命名空间；它只确保导入了包 sound.effects （可能运行任何在 \_\_init\_\_.py 中的初始化代码），然后导入包中定义的任何名称。这包括 \_\_init\_\_.py` 定义的任何名称（以及显式加载的子模块）。它还包括由之前的 [import](https://docs.python.org/zh-cn/3/reference/simple_stmts.html#import) 语句显式加载的包的任何子模块。思考下面的代码:

**import** **sound.effects.echo**

**import** **sound.effects.surround**

**from** **sound.effects** **import** \*

在这个例子中， echo 和 surround 模块是在执行 from...import 语句时导入到当前命名空间中的，因为它们定义在 sound.effects 包中。（这在定义了 \_\_all\_\_ 时也有效。）

虽然某些模块被设计为在使用 import \* 时只导出遵循某些模式的名称，但在生产代码中它仍然被认为是不好的做法。

请记住，使用 from package import specific\_submodule 没有任何问题！ 实际上，除非导入的模块需要使用来自不同包的同名子模块，否则这是推荐的表示法。

### 6.4.2. 子包参考

当包被构造成子包时（与示例中的 sound 包一样），你可以使用绝对导入来引用兄弟包的子模块。例如，如果模块 sound.filters.vocoder 需要在 sound.effects 包中使用 echo 模块，它可以使用 from sound.effects importecho 。

你还可以使用import语句的 from module import name 形式编写相对导入。这些导入使用前导点来指示相对导入中涉及的当前包和父包。例如，从 surround 模块，你可以使用:

**from** **.** **import** echo

**from** **..** **import** formats

**from** **..filters** **import** equalizer

请注意，相对导入是基于当前模块的名称进行导入的。由于主模块的名称总是 "\_\_main\_\_" ，因此用作Python应用程序主模块的模块必须始终使用绝对导入。

### 6.4.3. 多个目录中的包

包支持另一个特殊属性， [\_\_path\_\_](https://docs.python.org/zh-cn/3/reference/import.html#__path__) 。它被初始化为一个列表，其中包含在执行该文件中的代码之前保存包的文件 \_\_init\_\_.py 的目录的名称。这个变量可以修改；这样做会影响将来对包中包含的模块和子包的搜索。

虽然通常不需要此功能，但它可用于扩展程序包中的模块集。

**脚注**

|  |  |
| --- | --- |
| [[1]](https://docs.python.org/zh-cn/3/tutorial/modules.html#id1) | 实际上，函数定义也是“被执行”的“语句”；模块级函数定义的执行在模块的全局符号表中输入该函数名。 |