模块

1.字节编译的.pyc 文件

输入一个模块相对来说是一个比较费时的事情，所以 Python 做了一些技巧，以便使输入模块更加快一些。一种方法是创建 字节编译的文件 ，这些文件以`.pyc`作为扩展名。字节编译的文件与 Python 变换程序的中间状态有关（是否还记得 Python 如何工作的介绍？）。当你在下次从别的程序输入这个模块的时候，`.pyc`文件是十分有用的——它会快得多，因为一部分输入模块所需的处理已经完成了。另外，这些字节编译的文件也是与平台无关的。所以，现在你知道了那些`.pyc`文件事实上是什么了。

2.from..import 语句

如果你想要直接输入`argv`变量到你的程序中（避免在每次使用它时打`sys.`），那么你可以使用`from sys import argv`语句。如果你想要输入所有`sys`模块使用的名字，那么你可以使用`from sys import *`语句。这对于所有模块都适用。一般说来，应该避免使用`from..import`而使用`import`语句，因为这样可以使你的程序更加易读，也可以避免名称的冲突。

3.模块的 name

每个模块都有一个名称，在模块中可以通过语句来找出模块的名称。这在一个场合特别有用——就如前面所提到的，当一个模块被第一次输入的时候，这个模块的主块将被运行。假如我们只想在程序本身被使用的时候运行主块，而在它被别的模块输入的时候不运行主块，我们该怎么做呢？这可以通过模块的 **name** 属性完成。

```
#!/usr/bin/python
# Filename: using_name.py

if __name__ == '__main__':
    print 'This program is being run by itself'
else:
    print 'I am being imported from another module'
```

输出

```
$ python using_name.py
```

```
This program is being run by itself
$ python
>>> import using_name
I am being imported from another module
>>>
```

它如何工作

每个 Python 模块都有它的`__name__`，如果它是`'__main__'`，这说明这个模块被用户单独运行，我们可以进行相应的恰当操作。

4.制造你自己的模块

创建你自己的模块是十分简单的，你一直在这样做！每个 Python 程序也是一个模块。你已经确保它具有`.py`扩展名了。下面这个例子将会使它更加清晰。

```
#!/usr/bin/python
# Filename: mymodule.py

def sayhi():
    print 'Hi, this is mymodule speaking.'

version = '0.1'

# End of mymodule.py
```

上面是一个 模块 的例子。你已经看到，它与我们普通的 Python 程序相比并没有什么特别之处。我们接下来将看看如何在我们别的 Python 程序中使用这个模块。

记住这个模块应该被放置在我们输入它的程序的同一个目录中，或者在`sys.path`所列目录之一。

```
#!/usr/bin/python
# Filename: mymodule_demo.py

import mymodule

mymodule.sayhi()
print 'Version', mymodule.version
```

输出

```
$ python mymodule_demo.py
Hi, this is mymodule speaking.
Version 0.1
```

它如何工作

注意我们使用了相同的点号来使用模块的成员。Python 很好地重用了相同的记号来，使我们这些 Python 程序员不需要不断地学习新的方法。

下面是一个使用 `from..import` 语法的版本。

```
#!/usr/bin/python
# Filename: mymodule_demo2.py

from mymodule import sayhi, version
# Alternative:
# from mymodule import *

sayhi()
print 'Version', version
```

`mymodule_demo2.py` 的输出与 `mymodule_demo.py` 完全相同。

5.dir()函数

你可以使用内建的 `dir` 函数来列出模块定义的标识符。标识符有函数、类和变量。
当你为 `dir()`提供一个模块名的时候，它返回模块定义的名称列表。如果不提供参数，它返回当前模块中定义的名称列表。

```
$ python
>>> import sys
>>> dir(sys) # get list of attributes for sys module
['__displayhook__', '__doc__', '__excepthook__', '__name__', '__stderr__',
'__stdin__', '__stdout__', '_getframe', 'api_version', 'argv',
'builtin_module_names', 'byteorder', 'call_tracing', 'callstats',
'copyright', 'displayhook', 'exc_clear', 'exc_info', 'exc_type',
'excepthook', 'exec_prefix', 'executable', 'exit', 'getcheckinterval',
'getdefaultencoding', 'getdlopenflags', 'getfilesystemencoding',
'getrecursionlimit', 'getrefcount', 'hexversion', 'maxint', 'maxunicode',
'meta_path','modules', 'path', 'path_hooks', 'path_importer_cache',
'platform', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setdlopenflags',
'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout',
'version', 'version_info', 'warnoptions']
>>> dir() # get list of attributes for current module
['__builtins__', '__doc__', '__name__', 'sys']
>>>
>>> a = 5 # create a new variable 'a'
>>> dir()
['__builtins__', '__doc__', '__name__', 'a', 'sys']
>>>
>>> del a # delete/remove a name
```

```
>>>
>>> dir()
['__builtins__', '__doc__', '__name__', 'sys']
>>>
```

它如何工作

首先，我们来看一下在输入的 `sys` 模块上使用 `dir`。我们看到它包含一个庞大的属性列表。

接下来，我们不给 `dir` 函数传递参数而使用它——默认地，它返回当前模块的属性列表。注意，输入的模块同样是列表的一部分。

为了观察 `dir` 的作用，我们定义一个新的变量 `a` 并且给它赋一个值，然后检验 `dir`，我们观察到在列表中增加了以上相同的值。我们使用 `del` 语句删除当前模块中的变量/属性，这个变化再一次反映在 `dir` 的输出中。

关于 `del` 的一点注释——这个语句在运行后被用来 删除 一个变量/名称。在这个例子中，`del a`，你将无法再使用变量 `a`——它就好像从来没有存在过一样。