

模块

一、实验介绍

1.1 实验内容

在这个实验我们将要学习 Python 模块相关知识。

1.2 实验知识点

- 模块的导入
- 包
- 默认/第三方模块介绍
- 命令行参数

1.3 实验环境

- python3.5
- Xfce终端
- Vim

1.4 适合人群

本课程属于初级级别课程，不仅适用于那些有其它语言基础的同学，对没有编程经验的同学也非常友好

二、实验步骤

2.1 模块介绍

到目前为止，我们在 Python 解释器中写的所有代码都在我们退出解释器的时候丢失了。但是当人们编写大型程序的时候他们会倾向于将代码分为多个不同的文件以便使用，调试以及拥有更好的可读性。在 Python 中我们使用模块来到达这些目的。模块是包括 Python 定义和声明的文件。文件名就是模块名加上 `.py` 后缀。

你可以由全局变量 `__name__` 得到模块的模块名（一个字符串）。

现在来看看模块是怎样工作的。创建一个 `bars.py` 文件。文件内容如下：

```
"""
Bars Module
=====
这是一个打印不同分割线的示例模块
"""
def starbar(num):
    """打印 * 分割线

    :arg num: 线长
    """
    print('*' * num)

def hashbar(num):
    """打印 # 分割线

    :arg num: 线长
    """
    print('#' * num)

def simplebar(num):
    """打印 - 分割线

    :arg num: 线长
    """
    print('-' * num)
```

现在我们启动解释器然后导入我们的模块。

```
>>> import bars
>>>
```

我们必须使用模块名来访问模块内的函数。

```
>>> bars.hashbar(10)
#####
>>> bars.simplebar(10)
-----
>>> bars.starbar(10)
*****
```

2.2 导入模块

有不同的方式导入模块。我们已经看到过一种了。你甚至可以从模块中导入指定的函数。这样做：

```
>>> from bars import simplebar, starbar
>>> simplebar(20)
-----
```


你也可以使用 `from module import *` 导入模块中的所有定义，然而这并不是推荐的做法。

2.3 包

含有 `__init__.py` 文件的目录可以用来作为一个包，目录里的所有 `.py` 文件都是这个包的子模块，如下：

```
Terminal 终端 - shiyanlou@661b3c1c9b16: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ tree mymodule
mymodule
├── bars.py
├── __init__.py
└── utils.py
0 directories, 3 files
shiyanlou:~/ $
```

[15:08:41]

 实验楼
[15:08:44]

在这个例子中，`mymodule` 是一个包名并且 `bars` 和 `utils` 是里面的两个子模块。你可以使用 `touch` 命令创建一个空的 `__init__.py` 文件。

```
$ touch mymodule/__init__.py
```

如果 `__init__.py` 文件内有一个名为 `__all__` 的列表，那么只有在列表内列出的名字将会被公开。

因此如果 `mymodule` 内的 `__init__.py` 文件含有以下内容：

```
from mymodule.bars import simplebar
__all__ = [simplebar, ]
```

那么导入时将只有 `simplebar` 可用。

`from mymodule import *` 只能工作在模块级别的对象上，试图导入函数或类将导致 `syntax error`。

参考资料 (<https://docs.python.org/3/tutorial/modules.html#packages>)

2.4 默认模块

现在你安装 Python 的时候会附带安装不同的模块，你可以按需使用它们，也可以为其它特殊用途安装新模块。在下面的几个例子中，我们将要看到同样例子很多。

```
Terminal 终端 - python3
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
>>> help()

Welcome to Python 3.5's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/3.5/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help>

You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)". Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>> modules
```

上面的例子展示了怎样获得你系统中安装的所有模块的列表。在这里就不粘贴它们了，因为这是一个很大的列表。

你也能在解释器里使用 `help()` 函数查找任何模块/类的文档。如果你想要知道字符串所有可用的方法，你可以像下面这样做：

```
>>> help(str)
```

2.4.1 os 模块

`os` (<http://docs.python.org/3/library/os.html#module-os>) 模块提供了与操作系统相关的功能。你可以使用如下语句导入它：

```
>>> import os
```

`getuid()` 函数返回当前进程的有效用户 id。

```
>>> os.getuid()
500
```

`getpid()` 函数返回当前进程的 id。 `getppid()` 返回父进程的 id。

```
>>> os.getpid()
16150
>>> os.getppid()
14847
```

`uname()` 函数返回识别操作系统的不同信息，在 Linux 中它返回的详细信息可以从 `uname -a` 命令得到。 `uname()` 返回的对象是一个元组，（`sysname`, `nodename`, `release`, `version`, `machine`）。

```
>>> os.uname()
('Linux', 'd80', '2.6.34.7-56.fc13.i686.PAE', '#1 SMP Wed Sep 15 03:27:15 UTC 2010', 'i686')
```

`getcwd()` 函数返回当前工作目录。 `chdir(path)` 则是更改当前目录到 `path`。在例子中我们首先看到当前工作目录是 `/home/shiyanlou`，然后我们更改当前工作目录到 `/Code` 并再一次查看当前工作目录。

```
>>> os.getcwd()
'/home/shiyanlou'
>>> os.chdir('Code')
>>> os.getcwd()
'/home/shiyanlou/Code'
```

所以现在让我们使用 `os` 模块提供的另一个函数来创建一个自己的函数，它将列出给定目录下的所有文件和目录。

```
def view_dir(path='.'):
    """
    这个函数打印给定目录中的所有文件和目录
    :args path: 指定目录，默认为当前目录
    """
    names = os.listdir(path)
    names.sort()
    for name in names:
        print(name, end = ' ')
    print()
```

使用例子中的 `view_dir()` 函数。

```
>>> view_dir('/')  
.bashrc .dockerenv .profile bin boot dev etc home lib lib64 media mnt  
opt proc root run sbin srv sys tmp usr var
```

os 模块还有许多非常有用的函数，你可以在这里 (<https://docs.python.org/3/library/os.html>) 阅读相关内容。

2.5 Requests 模块

Requests (http://docs.python-requests.org/zh_CN/latest/) 是一个第三方 Python 模块，其官网的介绍如下：

Requests 唯一的一个非转基因的 Python HTTP 库，人类可以安全享用。

警告：非专业使用其他 HTTP 库会导致危险的副作用，包括：安全缺陷症、冗余代码症、重新发明轮子症、啃文档症、抑郁、头疼、甚至死亡。

第三方模块并不是默认模块，意味着你需要安装它，我们使用 pip3 安装它。

首先要安装 pip3：

```
$ sudo apt-get update  
$ sudo apt-get install python3-pip
```

然后用 pip3 安装 requests

```
$ sudo pip3 install requests
```

上面的命令会在你的系统中安装 Python3 版本的 Requests 模块。

2.5.1 获得一个简单的网页

你可以使用 `get()` 方法获取任意一个网页。

```
>>> import requests
>>> req = requests.get('http://www.baidu.com')
#非会员用户不能访问外网，所以请在学习的时候将url更换为 github.com 进行测试
>>> req.status_code
200
```

req 的 text 属性存有服务器返回的 HTML 网页，由于 HTML 文本太长就不在这里贴出来了。

使用这个知识，让我们写一个能够从指定的 URL 中下载文件的程序。

```
#!/usr/bin/env python3
import os
import os.path
import requests

def download(url):
    '''从指定的 URL 中下载文件并存储到当前目录

    :arg url: 要下载的文件 URL
    ...

    req = requests.get(url)
    # 首先我们检查是否存在文件
    if req.status_code == 404:
        print('No such file found at %s' % url)
        return
    filename = url.split('/')[-1]
    with open(filename, 'wb') as fobj:
        fobj.write(req.content)
    print("Download over.")

if __name__ == '__main__':
    url = input('Enter a URL: ')
    download(url)
```

测试一下程序：


```
Terminal 终端 · shiyanlou@661b3c1c9b16: ~
文件(F) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ ./download.py [15:22:07]
Enter a URL: http://labfile.oss.aliyuncs.com/courses/596/sample.txt
Download over.
shiyanlou:~/ $ ll [15:22:13]
总用量 16K
drwxrwxr-x 3 shiyanlou shiyanlou 4.0K 8月 17 13:52 Code
drwxrwxr-x 2 shiyanlou shiyanlou 4.0K 11月 27 2015 Desktop
-rwxrwxr-x 1 shiyanlou shiyanlou 502 8月 17 15:13 download.py
-rw-rw-r-- 1 shiyanlou shiyanlou 33 8月 17 15:22 sample.txt
shiyanlou:~/ $ [15:22:15]
```

可以看到目录下已经多了一个 sample.txt 文件。

你可能已经注意到了 `if __name__ == '__main__':` 这条语句，它的作用是，只有在当前模块名为 `__main__` 的时候（即作为脚本执行的时候）才会执行此 `if` 块内的语句。换句话说，当此文件以模块的形式导入到其它文件中时，`if` 块内的语句并不会执行。

你可以将上面的程序修改的更友好写。举个例子，你可以检查当前目录是否已存在相同的文件名。 `os.path`
(<http://docs.python.org/3/library/os.path.html#module-os.path>) 模块可以帮助你完成这个。

2.6 命令行参数

你还记得 `ls` 命令吗，你可以传递不同的选项作为命令行参数。你也可以在你的程序里通过 `argparse` 模块做到这点，阅读这篇 文档
(<https://docs.python.org/3/howto/argparse.html>) 学习。

2.7 TAB 补全

首先创建一个文件：`~/.pythonrc`，文件内写入如下内容：

```
import rlcompleter, readline
readline.parse_and_bind('tab: complete')

history_file = os.path.expanduser('~/.python_history')
readline.read_history_file(history_file)

import atexit
atexit.register(readline.write_history_file, history_file)
```

下一步在 `~/.bashrc` 文件中设置 `PYTHONSTARTUP` 环境变量指向这个文件：

```
$ export PYTHONSTARTUP=~/.pythonrc
```

现在，从今以后每当你打开 `bash shell`，你将会有 `TAB` 补全和 `Python` 解释器中代码输入的历史记录。

要在当前 `shell` 中使用，`source` 这个 `bashrc` 文件。

```
$ source ~/.bashrc
```

三、总结

本实验了解了什么是模块，模块怎样导入，举例了 `os` 和 `Requests` 模块的使用。`Python` 吸引人的一点是其有众多的模块可以使用，对于自带模块，可以看看 `Python3` 的官方文档 (<https://docs.python.org/3/library/index.html>)，对于第三方模块，可以在 `PyPI` (<https://pypi.python.org/pypi>) 上找找。很多时候你都能找到合适的包帮你优雅的完成部分工作。比如 `argparse` 模块帮你非常容易的编写用户友好的命令行接口。

**本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

上一节：类 (/courses/596/labs/2046/document)

下一节：Collections 模块 (/courses/596/labs/2048/document)

