

## 第七章 模块

7.1 模块的概述

7.2 安装第三方模块

7.3 模块应用实例

7.4 在Python中调用R语言

7.5 实验

7.6 小结

7.7 习题

### 7.1.1 模块与程序

我们写的代码并以.py结尾保存的Python文件就是一个独立的模块，模块包含了对对象定义和语句。如下所示代码保存至文件my\_module.py：

```
def fibonacci(n):  
    x3= 1  
    x1, x2= 1,1  
    if n < 1 :  
        print('输入有误! ')  
        return -1  
    while n > 2:  
        x3 = x2 + x1  
        x1, x2 = x2, x3  
        n -= 1  
    return x3
```

my\_module.py就是一个模块，其中定义了一个函数。

模块就是一个以.py结尾的独立的程序代码的文件，实现了特定的功能。通常，在模块中定义类、函数或常量，以便代码复用。

注意，如果模块中含有类或函数之外的代码，则导入模块时，这些代码将被执行。

### 7.1.2 命名空间

命名空间是一个包含了一个或多个变量名称和它们各自对应的对象值的字典。

Python可以调用局部命名空间和全局命名空间里的变量。如果一个局部变量和一个全局变量重名，则在函数内部调用时局部变量会屏蔽全局变量。

如果要修改函数内的全局变量的值，必须使用global语句，否则会出错。

### 7.1.3 模块导入方法

要导入系统模块或者已经定义好的模块，有三种方法：

1、最常用的方法是：

#### import module

module——是**模块名**，如果有多个模块，模块名称之间用逗号“,”隔开，但通常每一个模块单独使用import语句导入。导入模块后，就可以**引用模块内的函数**，语法格式：**模块名.函数名**

例如：

```
import random # 导入系统模块
```

```
import my_module # 导入自定义模块
```

```
number = random.randint(3, 20) # 产生一个随机整数
```

```
result = my_module.fibonacci(number)
```

```
print("斐波那契数列第%d项是： %d" % (number, result))
```

某两次的运行结果：

斐波那契数列第9项是： 34

斐波那契数列第13项是： 233

### 7.1.3 模块导入方法

#### 注意事项:

- (1)在IDLE交互环境中，有一个使用的小技巧，当输入导入的模块名和点号 “.” 之后，系统会将模块内的函数罗列出来供我们选择。
- (2)可以通过help(模块名)查看模块的帮助信息，其中，FUNCTIONS介绍了模块内函数的使用方法。
- (3)不管你执行了多少次import，一个模块只会被导入一次。
- (4)导入模块后，我们就可用模块名称这个变量访问模块的函数等所有功能。

### 7.1.3 模块导入方法

2、第二种方法是：

**from 模块名 import 函数名**

函数名如果有多个，可用逗号 “,” 隔开。

函数名可用通配符 “\*” 导出所有的函数。

这种方法要**慎用**，因为**导出的函数名称容易和其它函数名称冲突**，失去了模块命名空间的**优势**。

**例如：**

**from** random **import** randint # 导入系统模块中函数

**from** my\_module **import** fibonacci # 导入自定义模块中函数

number = randint(3, 20) # 产生一个随机整数

result = fibonacci(number)

print("斐波那契数列第%d项是： %d" % (number, result))

### 7.1.3 模块导入方法

3、第三种方法是：

**import 模块名 as 新名字**

这种导入模块的方法，相当于给导入的模块名称重新起一个**别名**，便于记忆，很方便地在程序中调用。

注意，这种情况下，原先的模块名称就不能使用了，而要使用新名字。

例如：

```
import random as rnd # 导入系统模块并取别名为rnd
import my_module # 导入自定义模块
```

```
number = rnd.randint(3, 20) # 产生一个随机整数
result = my_module.fibonacci(number)
print("斐波那契数列第%d项是： %d" % (number, result))
```

**random.randint(3, 20)**则编译出错  
"Unresolved reference 'random' "

### 7.1.4 自定义模块和包

#### 1.自定义模块：

自定义模块的方法和步骤如下：

在安装Python的目录下，新建一个以.py为后缀名的文件，然后编辑该文件。

**在自定义模块时，有几点要注意：**

- (1)为了使IDLE能找到我们自定义模块，该模块要和调用的程序在同一目录下，否则在导入模块时会提示找不到模块的错误。
- (2)模块名要遵循Python变量命名规范，不要使用中文、特殊字符等。
- (3)自定义的模块名不要和系统内置的模块名相同，可以先在IDLE 交互环境里先用 “import modle\_name”命令检查，若成功则说明系统已存在此模块，然后考虑更改自定义的模块名。



### 7.1.4 自定义模块和包

#### 2.自定义包:

在大型项目开发中，有多个程序员协作共同开发一个项目，为了避免模块名重名，Python引入了按目录来组织模块的方法，称为**包（Package）**。**包是一个分层级的文件目录结构，它定义了由模块及子包，以及子包下的子包等组成的命名空间。**

**在自定义包时，需要注意：**

- (1)每个包目录下面都会有一个\_\_init\_\_.py的文件，**这个文件是必须存在的**，否则，系统就把这个目录作为普通目录，而不是一个包。
- (2)\_\_init\_\_.py可以是空文件(默认即为空文件)，也可以有Python代码，因为\_\_init\_\_.py就是一个模块，而它的**模块名就是mymodule**。
- (3)在Python中可以有多级目录，组成多层次的包结构。

### 包中\_\_init\_\_.py的作用

- Python中package的标识，不能删除。每个package实际上是一个目录（Directory），如果没有\_\_init\_\_.py文件，该目录就不会认为是package。
- 定义\_\_all\_\_用来模糊导入。

Python中的包和模块有两种导入方式：精确导入和模糊导入。

■ 精确导入：`from myPack import module1, module2`

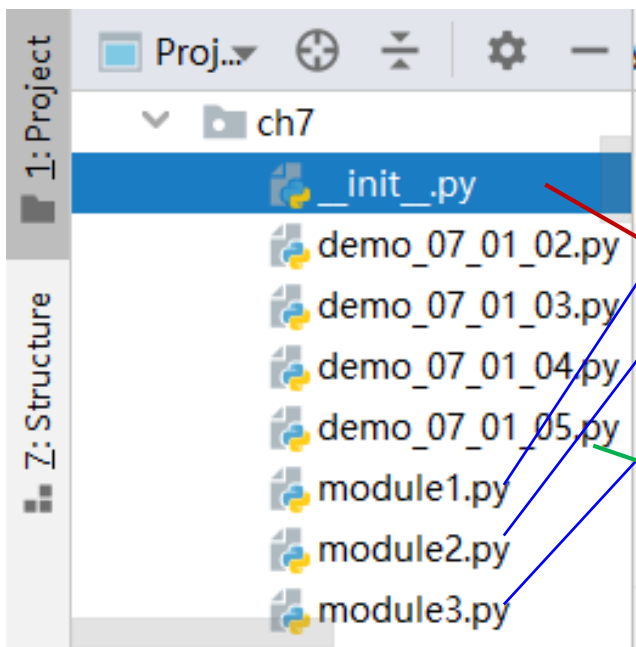
■ 模糊导入：`from myPack import *`

模糊导入中的\*中的模块是由全局变量\_\_all\_\_来定义的，如在\_\_init\_\_.py中编写代码：

```
__all__ = [" module1 ", " module2 "]
```

- 调用包的时候首先会执行\_\_init\_\_.py文件。
- 编写Python代码，但不建议在\_\_init\_\_中写python模块，我们可以在包中在创建另外的模块来写，尽量保证\_\_init\_\_.py简单。

### 使用\_\_init\_\_.py示例：定义模糊导入



```
def f():  
    print('f() in module1 called.')
```

```
def f():  
    print('f() in module2 called.')
```

```
def f():  
    print('f() in module3 called.')
```

```
___all___ = ['module1', 'module2'] # 定义模糊导入
```

```
from ch7 import * # 模糊导入
```

```
module1.f()  
module2.f()  
module3.f() # Unresolved reference 'module3'
```

#### 运行结果：

f() in module1 called.

f() in module2 called.

Traceback (most recent call last):

File "C:/Users/ruanz/PycharmProjects/new/ch7/demo\_07\_01\_05.py", line 16, in <module>  
 module3.f() # Unresolved reference 'module3',  
NameError: name 'module3' is not defined

### 7.1.5 引用包和包中模块

如果模块定义在包中，则使用时需要导入包，有两种方式：

- `from 包 import 模块 [as 新名字]`

导入包中的某个模块，推荐采用的方式。可以给模块取新名字。

- `import 包.模块 [as 新名字]`

导入包中的某个模块。建议给模块取新名字，否则调用模块中的函数时，需要使用包名限定，形如 `包.模块.函数 (...)`。

**示例：**假定在包new中创建了包ch7，且其中定义了模块my\_module

```
import my_module # 使用同一个包中的模块，直接载入该模块即可
```

```
# import new.ch7.my_module # 从最顶层的包开始载入，也可以
```

```
# import new.ch7.my_module as my_module
```

```
result = my_module.fibonacci(6)
```

调用程序所在包中模块的导入方式

### 7.1.5 引用包和包中模块

如果模块定义在包中，则使用时需要导入包，有两种方式：

- `from 包 import 模块 [as 新名字]`

导入包中的某个模块，推荐采用的方式。可以给模块取新名字。

- `import 包.模块 [as 新名字]`

导入包中的某个模块。建议给模块取新名字，否则调用模块中的函数时，需要使用包名限定，形如 `包.模块.函数 (...)`。

**示例：**包new/ch6创建了某个模块demo\_xxx.py，在其中使用ch7包中的my\_module模块

```
import sys
sys.path.append(r'../ch7') # 将ch7包添加到包搜索路径集合中
import my_module # 导入ch7中的my_module

result = my_module.fibonacci(6)
```

调用程序所在  
包之外的模块  
的导入方式

### 7.1.5 引用包和包中模块

**示例：**包new/ch6创建了模块demo\_07\_01\_05.py，在其中使用ch7包内的模块module1、module2、module3

```
import sys
sys.path.append(r'../') # 将ch7包的所在的目录，即本模块文件所在目录的上  
一层目录加入系统路径列表
```

```
# from ch7 import module1, module2
```

```
from ch7 import *
```

等价

调用程序所在  
包之外的模块  
的导入方式

```
module1.f()
```

```
module2.f()
```

```
module3.f() # Unresolved reference 'module3',  
# 因为module3未在ch7/__init__.py文件的__all__中定义
```

```
__all__ = ['module1','module2'] # 定义模糊导入*所导入的模块
```

### 使用包及其模块示例：使用matplotlib库绘图

```
import matplotlib.pyplot as plt # 导入绘图库，并重命名为plt
import random
import math

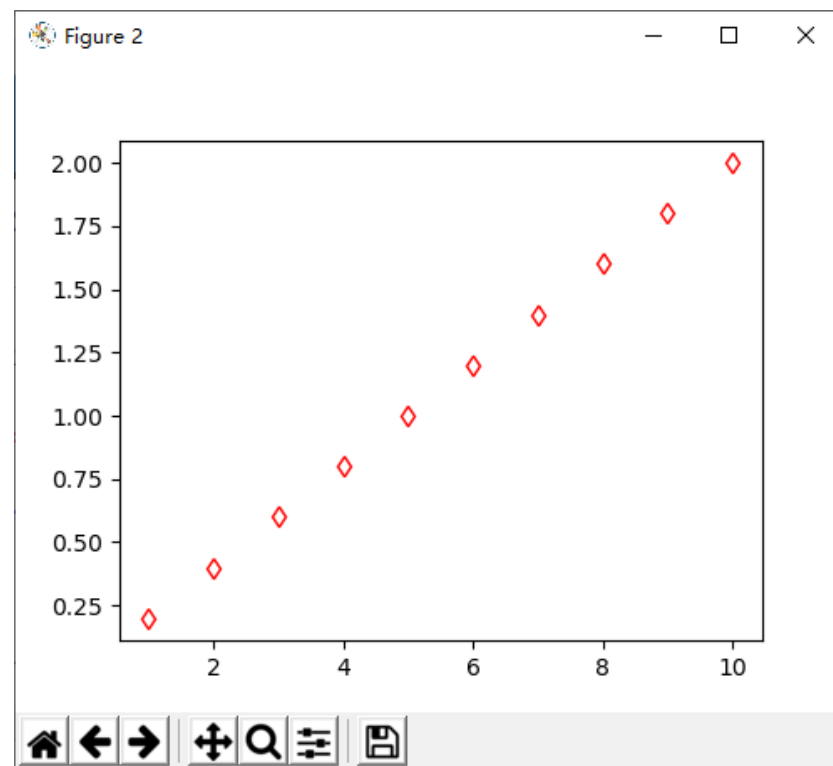
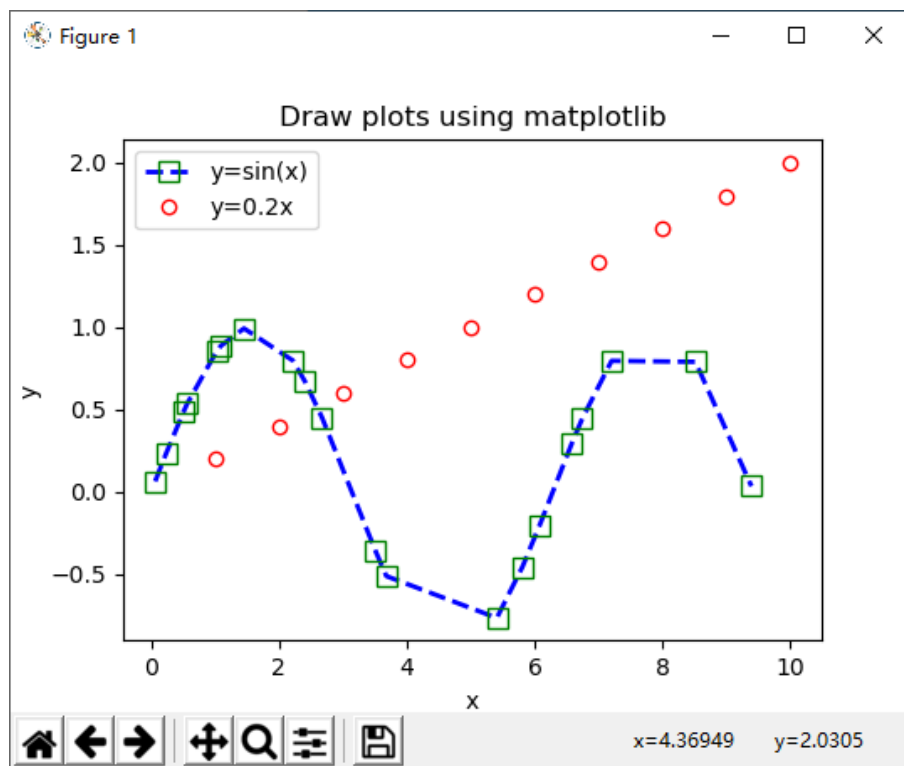
x = [random.uniform(0, 10) for i in range(0, 20)] # 推导式列表随机产生均匀分布的x坐标
x.sort() # 升序排列
y = [math.sin(xi) for xi in x] # 使用推导列表生成y=sin(x), <=> y=list(map(math.sin,x))
x2 = range(1, 11)
y2 = list(map(lambda t: 0.2 * t, x2))

plt.figure() # 新建一个绘图窗口
plt.plot(x, y, '--sb', lw=2, markersize=8, mec='g', mfc='none', label='y=sin(x)') # 绘制曲线
plt.plot(x2, y2, 'o', color='#FF0000', markerfacecolor='none', label='y=0.2x')
plt.legend() # 显示图例
plt.title("Draw plots using matplotlib")
plt.xlabel('x')
plt.ylabel('y')

plt.figure() # 新建另一个图形窗口
plt.plot(x2, y2, 'd', color='#FF0000', mfc='none', label='y=0.2x')

plt.show() # 显示图形窗体
```

## 使用包及其模块示例：使用matplotlib库绘图





## 第七章 模块

7.1 模块的概述

7.2 安装第三方模块

7.3 模块应用实例

7.4 在Python中调用R语言

7.5 实验

7.6 小结

7.7 习题

安装第三方模块，是通过包管理工具pip来实现的。

本节以Win10操作系统，Python 3.8.3安装为例，确保安装时勾选了pip和Add Python to environment variables两个选项。

在“开始” —>“运行” 里输入 “cmd”命令或者直接选中 “命令提示符” 。

pip命令格式如下:

```
pip <command> [options]
```

commands:

install        Install packages.

download      Download packages.

uninstall     Uninstall packages.

freeze        Output installed packages in requirements format.

.....

安装第三方模块前的注意事项：

(1)确保可以从命令提示符中的命令行运行Python。

请确保安装有Python，并且预期的版本可以从命令行获得，可以通过运行以下命令来检查：

```
python --version
```

运行结果如下：

```
C:\Users\Administrator>python --version
```

```
Python 3.8.3
```

(2)确保可以从命令行运行pip。

此外，还需要确保系统有pip可用，可以通过运行以下命令来检查：

```
pip --version
```

运行结果如下：

```
C:\Users\Administrator>pip --version
```

```
pip 20.1.1 from C:\Users\ruanz\AppData\Roaming\Python\Python38\site-  
packages\pip (python 3.8)
```

(3)确保pip、setuptools和wheel是最新的。

虽然pip单独地从预构建的二进制文件中安装就可以了，但是最新的setuptools和wheel的版本对于确保你也可以从源文件中安装是有用的。

可以运行以下命令：

```
python -m pip install --upgrade pip setuptools wheel
```

运行成功后得到，会有如下提示信息（如果全部都有升级更新）：

```
Successfully installed pip-20.1.1 setuptools-49.1.0 wheel-0.34.2
```

(4)创建一个虚拟环境，此项仅用于Linux系统，为可选项。运行以下命令：

```
python3 -m venv tutorial_env
```

```
source tutorial_env/bin/activate
```

上述命令将在tutorial\_env子目录中创建一个新的虚拟环境，并配置当前shell以将其用作默认的Python环境。

本节仅以从Python Package Index (PyPI)安装为例，其它安装方式请查阅相关资料。

使用pip从PyPI安装：

pip最常用的用法是从Python包索引中使用需求说明符来安装。一般来说，需求说明符由项目名称和版本说明符组成。

在Python官网<https://www.pypi.org>可以查询、注册、发布的第三方库，包括包的历史版本号，支持的应用环境等包信息。例如，matplotlib库。

[Help](#)[Sponsor](#)[Log in](#)[Register](#)

Filter by [classifier](#)

2,821 projects for "matplotlib"

Order by

Relevance

☒ Framework

☒ Topic



**matplotlib 3.2.2**

Python plotting package

Jun 18, 2020

我们以安装web模块为例：

(1)在Python官网查询：web，得到包的名称是：web3，最新版本号是：4.3.0。

在命令提示符下输入以下命令：

```
pip install web3==4.3.0
```

系统自动会从Python官网下载文件，进行安装。

在安装过程中，有的系统环境也许会出现以下错误提示：

```
error: Microsoft Visual C++ 14.0 is required. Get it with "Microsoft  
Visual C++ Build Tools": http://landinghub.visualstudio.com/  
visual-cpp-build-tools
```

解决办法是：

下载：visualcppbuildtools\_full.exe安装即可。



(2)升级包：

将已安装的项目升级到PyPI的最新项目，通过运行以下命令：

```
pip install --upgrade web3
```

(3)安装到用户站点

若要安装与当前用户隔离的包，请使用用户标志，通过运行以下命令：

```
pip install --user SomeProject
```

(4)需求文件：

安装需求文件中指定的需求列表，如果没有则忽略。通过运行以下命令：

```
pip install -r requirements.txt
```

(5)在Python shell环境中验证安装的第三方模块：

在IDLE Shell交互环境下使用import命令，如下所示：

```
>>> import web3
```

运行结果如下：

```
>>> dir(web3)
```

```
['Account', 'EthereumTesterProvider', 'HTTPProvider', 'IPCProvider',  
'TestRPCProvider', 'Web3', 'WebsocketProvider', '__all__', '__builtins__',  
'__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',  
'__path__', '__spec__', '__version__', 'admin', 'contract', 'eth', 'exceptions',  
'iban', 'main', 'manager', 'middleware', 'miner', 'module', 'net', 'parity', 'personal',  
'pkg_resources', 'providers', 'sys', 'testing', 'txpool', 'utils', 'version']
```

从以上运行结果可以看出，第三方模块web已成功安装。

## 第七章 模块

7.1 模块的概述

7.2 安装第三方模块

7.3 模块应用实例

7.4 在Python中调用R语言

7.5 实验

7.6 小结

7.7 习题

### 7.3.1 日期时间相关：datetime模块

**datetime**是Python处理日期和时间的标准模块。

(1)获取当前日期和时间：

如下例所示代码：

```
>>> from datetime import datetime
>>> now = datetime.now() # 获取当前datetime
>>> now
datetime.datetime(2020, 7, 5, 16, 33, 11, 436199)
>>> print(now)
2020-07-05 16:33:11.436199
>>> '{:%Y-%m-%d}'.format(now) # 格式化日期
'2020-07-05'
>>> '{:%H:%M:%S}'.format(now) # 格式化时间
'16:33:11'
```

datetime是模块，其中还包含一个datetime类，通过from datetime import datetime导入的才是datetime这个类。如果仅导入import datetime，则必须引用全名datetime.datetime。  
datetime.now()返回当前日期和时间，其类型是datetime。

### 7.3.1 日期时间相关：datetime模块

(2)获取指定日期和时间：

如下所示代码：

```
>>> from datetime import datetime
```

```
>>> dt = datetime(2018, 6, 19, 13, 15) # 用指定日期时间创建datetime
```

运行结果如下：

```
>>> print(dt)
```

```
2018-06-19 13:15:00
```

### 7.3.1 日期时间相关：datetime模块

(3)datetime转换为timestamp:

在计算机中，时间实际上是用数字表示的。我们把1970年1月1日 00:00:00 UTC+00:00时区的时刻称为**新纪元时间(Epoch Time)**，记为0（1970年以前的时间timestamp为负数），当前时间就是相对于epoch time的秒数，称为timestamp。

UTC(Coordinated Universal Time)一般指协调世界时，又称世界统一时间、世界标准时间、国际协调时间。中国大陆、中国香港、中国澳门、中国台湾、蒙古国、新加坡、马来西亚、菲律宾、西澳大利亚州的时间与UTC的时差均为+8，也就是UTC+8。

你可以认为：timestamp = 0 = 1970-1-1 00:00:00 UTC+0:00

对应的北京时间是：

timestamp = 0 = 1970-1-1 08:00:00 UTC+8:00

### 7.3.1 日期时间相关：datetime模块

可见timestamp的值与时区毫无关系，因为timestamp一旦确定，其UTC时间就确定了，转换到任意时区的时间也是完全确定的，这就是为什么计算机存储的当前时间是以timestamp表示的，因为全球各地的计算机在任意时刻的timestamp都是完全相同的。

把一个datetime类型转换为timestamp只需要简单调用timestamp()方法，如下所示代码：

```
>>> from datetime import datetime
```

```
>>> dt = datetime(2020, 7, 5, 17, 26) # 用指定日期时间创建datetime
```

### 7.3.1 日期时间相关：datetime模块

运行结果如下：

```
>>> dt.timestamp() # 把datetime转换为timestamp
```

```
1593941160.0
```

**注意：**Python的timestamp是一个浮点数。如果有小数位，小数位表示毫秒数。某些编程语言（如Java和JavaScript）的timestamp使用整数表示毫秒数，这种情况下只需要把timestamp除以1000就得到Python的浮点表示方法。



### 7.3.1 日期时间相关：datetime模块

(4)timestamp转换为datetime:

要把timestamp转换为datetime，使用datetime提供的fromtimestamp()方法，如下所示代码：

```
>>> from datetime import datetime
```

```
>>> t = 1593941160.0
```

运行结果如下：

```
>>> print(datetime.fromtimestamp(t))
```

```
2020-07-05 17:26:00
```

### 7.3.1 日期时间相关：datetime模块

从上例可以看出，timestamp是一个浮点数，它没有时区的概念，而datetime是有时区的。上述转换是**在timestamp和本地时间做转换**。**本地时间**是指**当前操作系统设定的时区**。

timestamp也可以直接被转换到UTC标准时区的时间，使用datetime提供的utcfromtimestamp()方法，如下所示代码：

```
>>> from datetime import datetime
>>> t = 1593941160.0
>>> print(datetime.fromtimestamp(t)) # 本地时间
2020-07-05 17:26:00
>>> print(datetime.utcfromtimestamp(t)) # UTC时间
2020-07-05 09:26:00
```

### 7.3.1 日期时间相关：datetime模块

(5)str转换为datetime:

用户输入的日期和时间是字符串，要处理日期和时间，首先必须把str转换为datetime。转换方法是通过datetime提供的strptime()方法来实现，如下例所示代码：

```
>>> from datetime import datetime
>>> datee_test = datetime.strptime('2020-07-05 17:26:00', '%Y-%m-%d
%H:%M:%S')
>>> print(datee_test)
'2020-07-05 17:26:00'
```

其中，字符串'%Y-%m-%d %H:%M:%S'规定了日期和时间部分的格式。转换后的datetime是没有时区信息的。

### 7.3.1 日期时间相关：datetime模块

(6)datetime转换为str:

如果已经有了datetime对象，要把它格式化为字符串显示给用户，就需要转换为str，转换方法是通过datetime提供的strftime()方法实现的，如下例所示代码：

```
>>> from datetime import datetime
```

```
>>> now = datetime.now()
```

```
>>> print(now.strftime('%a, %b %d %H:%M'))
```

```
Sun, Jul 05 16:33
```

### 7.3.1 日期时间相关：datetime模块

(7)datetime加减：

对日期和时间进行加减，实际上就是把datetime往后或往前计算，得到新的datetime。加减可以直接用+和-运算符，需要导入**timedelta**类，例如：

```
>>> from datetime import datetime, timedelta
```

```
>>> now = datetime.now()
```

```
>>> now
```

```
datetime.datetime(2020, 7, 5, 18, 4, 41, 789407)
```

```
>>> now + timedelta(hours=10)
```

```
datetime.datetime(2020, 7, 6, 4, 4, 41, 789407)
```

```
>>> now - timedelta(days=10)
```

```
datetime.datetime(2020, 6, 25, 18, 4, 41, 789407)
```

```
>>> now + timedelta(days=12, hours=23)
```

```
datetime.datetime(2020, 7, 18, 17, 4, 41, 789407)
```

### 7.3.1 日期时间相关：datetime模块

(8)本地时间转换为UTC时间：

本地时间是指系统设定时区的时间，例如北京时间是UTC+8:00时区的时间，而UTC时间指UTC+0:00时区的时间。

datetime类型有时区属性tzinfo，默认为None，所以无法区分这个datetime到底是哪个时区，除非强行给datetime设置一个时区，如下例所示代码：

```
>>> from datetime import datetime, timedelta, timezone
```

```
>>> utc_8 = timezone(timedelta(hours=8)) # 创建时区UTC+8:00
```

```
>>> now = datetime.now()
```

### 7.3.1 日期时间相关：datetime模块

```
>>> now
```

```
datetime.datetime(2020, 7, 5, 18, 23, 43, 928403)
```

```
>>> dt_test = now.replace(tzinfo=utc_8) # 强制设置为UTC+8:00
```

```
>>> dt_test
```

```
datetime.datetime(2020, 7, 5, 18, 23, 43, 928403, tzinfo=datetime.timezone  
(datetime.timedelta(seconds=28800)))
```

从上例可以看出，如果系统时区恰好是UTC+8:00，那么上述程序代码正确，否则，不能强制设置为UTC+8:00时区。

### 7.3.1 日期时间相关：datetime模块

(9)时区转换：

先通过datetime提供的utcnow()方法拿到当前的UTC时间，再用astimezone()方法转换为任意时区的时间，如下例所示：

获取UTC时间，并强制设置时区为UTC+0:00: 如下所示代码：

```
>>> utc_dtime = datetime.utcnow().replace(tzinfo=timezone.utc)
```

```
>>> print(utc_dtime)
```

```
2020-07-05 10:55:53.103202+00:00
```

将转换时区为北京时间，如下所示代码：

```
>>> bj_dtime = utc_dtime.astimezone(timezone(timedelta(hours=8)))
```

```
>>> print(bj_dtime)
```

```
2020-07-05 18:55:53.103202+08:00
```

可见，时区转换的关键在于得到datetime时间，要获知其正确的时区，然后强制设置时区，作为基准时间。利用带时区的datetime，通过astimezone()方法，可以转换到任意时区。



### 7.3.2 读写JSON数据：json模块

JSON(JavaScript Object Notation) 是一种轻量级的**数据交换格式**。

**JSON的数据格式等同于Python里面的字典格式**，里面可以包含中括号括起来的数组，即python里面的列表。

很多动态网页的数据请求常常以JSON格式字符串返回，例如百度在线翻译的结果：

```
{  
  "errno": 0,  
  "data": [  
    {"k": "网页", "v": "名. web page"},  
    {"k": "网页制作", "v": "名. webpage making"},  
    {"k": "网页美工", "v": "名. Web Creative"},  
    {"k": "网页设计", "v": "名. Web Design"},  
    {"k": "网页设计师", "v": "名. web designer"} ]  
}
```

### 7.3.2 读写JSON数据：json模块

在python中，json模块专门处理JSON格式的数据与字典的相互转化，提供了四种方法：dumps、dump、loads、load。

(1) dumps、dump方法：

dumps、dump方法实现字典数据的序列化功能，即字典转化为JSON格式数据，其中：

- dumps方法实现的是将字典数据序列化为JSON格式的字符串(str)；
- 在使用dump方法时，必须传文件描述符，将字典数据以JSON格式保存到文本文件，这种格式的文本文件被称为JSON文件。

### 7.3.2 读写JSON数据：json模块

Dumps方法的使用，如下所示代码：

```
>>> import json
>>> json.dumps('Python') #字符串
'"Python"'
>>> json.dumps(12.78) #数字
'12.78'
>>> dict_test = {"university": "UPC", "nation_rank": 85} #字典
>>> json.dumps(dict_test) #字典
'{"university": "UPC", "nation_rank": 85}'
```

其中，dumps将数字、字符串、字典等数据序列化为标准的字符串(str)格式。

### 7.3.2 读写JSON数据：json模块

dump方法的使用，如下所示：

```
import json
```

```
dict_test = {"university": "UPC", "nation_rank": 85} # 字典
```

```
with open(r"c:\test\json_test.json", "w", encoding='utf-8') as file_test:  
    json.dump(dict_test, file_test, indent=4)
```

dump方法将字典数据dict\_test保存到c:\test\json\_test.json文件中：



### 7.3.2 读写JSON数据：json模块

(2) loads、load方法：

Loads、load是反序列化方法。loads 方法只完成了反序列化，即将**JSON格式的字符串**转化为**字典**，而load方法 只接收文件描述符，完成了读取**JSON文件并反序列化为字典**。

Loads方法的使用，如下所示代码：

```
>>> import json
```

```
>>> json.loads('{"university": "UPC", "nation_rank": 85}')
```

```
{"university": "UPC", "nation_rank": 85}
```

本例中，loads将已经序列化的字典字符串数据反序列化为字典数据。

### 7.3.2 读写JSON数据：json模块

Load方法的使用，如下所示代码：

```
import json
```

```
with open(r"c:\test\json_test.json", "r", encoding='utf-8') as file_test:
```

```
    dt= json.load(file_test)
```

```
print(type(dt))
```

```
print(dt)
```

```
<class 'dict'>
{'university': 'UPC', 'nation_rank': 85}
```

等价于：

```
with open(r"c:\test\json_test.json", "r", encoding='utf-8') as file_test:
```

```
    text = file_test.read()
```

```
    json_text = json.loads(text)
```

```
print(type(dt))
```

```
print(json_text)
```

load将已经序列化的文件的字典字符串数据反序列化为字典数据，loads实现了和load一样的功能。

### 7.3.3 系统相关：sys模块

sys模块是python自带模块，包含了和系统相关的信息。通过运行以下命令，导入该模块，如下所示代码：

```
>>> import sys
```

通过help(sys)或者dir(sys)命令查看sys模块可用的方法,如下所示代码：

```
>>> dir(sys)
```

```
['__displayhook__', '__doc__', '__excepthook__', '__interactivehook__',  
'__loader__', '__name__', '__package__', '__spec__', '__stderr__', '__stdin__',  
'__stdout__', '_clear_type_cache', '_current_frames', '_debugmallocstats',  
'_enablelegacywindowsfsencoding',.....]
```

以上命令，显示了sys模块可用的方法。

### 7.3.3 系统相关：sys模块

下面列举sys模块常用的几种方法：

(1)sys.path: 包含输入模块的目录名列表。

```
>>> sys.path
```

```
['C:\\Program Files\\JetBrains\\PyCharm Community Edition  
2020.1.1\\plugins\\python-ce\\helpers\\pydev',
```

```
...
```

```
'C:\\Program Files\\Python38\\lib',
```

```
'C:\\Program Files\\Python38',
```

```
'C:\\Users\\ruanz\\AppData\\Roaming\\Python\\Python38\\site-packages',
```

```
'C:\\Program Files\\Python38\\lib\\site-packages',
```

```
'C:\\Users\\ruanz\\PycharmProjects\\classTest',
```

```
'C:/Users/ruanz/PycharmProjects/classTest']
```

我们可以将写好的模块放在得到的某个路径下，就可以在程序中import时正确找到。在import导入模块名时，就是根据sys.path的路径来搜索模块名，也可以用命令**sys.path.append(“自定义模块路径”)**添加模块路径。



### 7.3.3 系统相关：sys模块

(2)sys.argv：在外部向程序内部传递参数。

因为我们从外部取得的参数可以是多个，sys.argv是一个列表，所以能用[ ]提取其中的元素。其第一个元素是程序本身，随后才依次是外部给予的参数。例如：

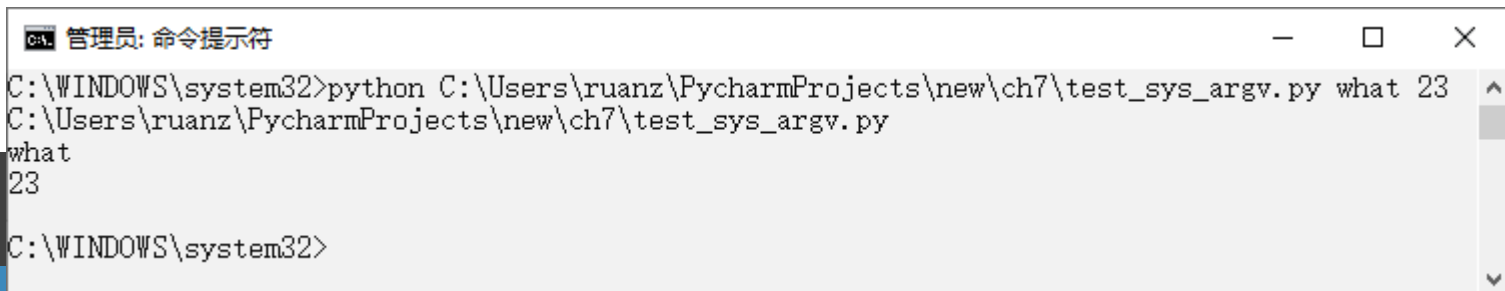
```
# test_sys_argv.py
```

```
import sys
```

```
for x in sys.argv: # 遍历命令行参数  
    print(x)
```

**在命令行执行本python程序，同时输入若干参数：**

```
C:\WINDOWS\system32>python C:\Users\ruanz\PycharmProjects\new\ch7\test_sys_argv.py what 23
```



```
管理员: 命令提示符  
C:\WINDOWS\system32>python C:\Users\ruanz\PycharmProjects\new\ch7\test_sys_argv.py what 23  
C:\Users\ruanz\PycharmProjects\new\ch7\test_sys_argv.py  
what  
23  
C:\WINDOWS\system32>
```

### 7.3.4 数学: math模块

math模块是python自带模块，包含了和数学运算公式相关的信息。

导入该模块：

```
>>> import math
```

通过dir(math)命令查看math模块可用的方法：

```
>>> dir(math)
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',  
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',  
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp',  
'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp',  
'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin',  
'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

上面的运行结果显示了math模块可用的函数。

### 7.3.5 随机数: random模块

random模块是python自带模块，功能是生成随机数。

导入该模块：

```
>>> import random
```

通过dir(random)命令查看random模块可用的方法：

```
>>> dir(random)
```

```
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random',  
'SG_MAGICCONST', 'SystemRandom', 'TWOPI', '_BuiltinMethodType',  
'_MethodType', '_Sequence', '_Set', '__all__', '__builtins__',  
'__cached__', .....]
```

### 7.3.5 随机数: random模块

下面，列举random模块部分常用的方法：

(1) 生成随机整数：randint()

运行以下代码，得到如下结果：

```
>>> random.randint(10,2390)
```

```
1233
```

注意：上例用于生成一个指定范围内的整数，其中**下限必须小于上限**，否则，程序会报错，如下面例子所示代码：

```
>>> random.randint(20,10)      #下限20 > 上限10
```

### 7.3.5 随机数:random模块

(2) 随机浮点数: random

运行以下代码，得到如下结果：

```
>>> random.random() #不带参数
```

```
0.47203863107027433
```

```
>>> random.uniform(35, 100) #带上限，下限参数
```

```
78.02991602825188
```

```
>>> random.uniform(350, 100)
```

```
232.2659504153889
```

从上例的运行结果可见，随机浮点数时，**下限可以大于上限。**

### 7.3.5 随机数:random模块

(3)随机字符: choice

```
>>> random.choice('98&@!~gho^')  
'g'
```

(4)洗牌: shuffle()

```
>>> test_shuffle = ['A','Q',1,6,7,9]  
>>> random.shuffle(test_shuffle)  
>>> test_shuffle  
[7, 6, 1, 'A', 'Q', 9]  
A', 'Q', 9]
```

(5)抽样: sample()

```
>>> test_color = ['Red','Green','Blue']  
>>> random.sample(test_color,1)  
['Blue']  
>>> random.sample(test_color,1)  
['Red']  
>>> random.sample(test_color,2)  
['Red', 'Blue']  
>>> random.sample(test_color,2)  
['Green', 'Red']
```

### 7.3.4 Faker库---生成伪造数据

在编写程序过程中，常常需要用到很多数据来进行测试。如果手动伪造数据，肯定要花费大把精力。此时，可以应该使用**第三方的Faker库**来生成各种各样的**伪数据**。

#### (1) 安装Faker

```
pip install Faker
```

#### (2) 基本用法

```
from faker import Faker # 从faker模块导入Faker类
```

```
fake = Faker() # Faker实例
```

```
print(fake.name()) # 调用name()方法随机生成一个名字
```

```
print(fake.address()) # 调用address()方法随机生成地址信息
```

某次运行的结果：

Jill Williamson

66098 Chavez Mountain

East Robert, AL 79396

另一次运行的结果：

Danielle Wood

498 Gilbert Course

South Marktown, MS 58035

### 7.3.4 Faker库---生成伪造数据

如果要生成简体中文的随机数据，可以在实例化时给locale参数传入'zh\_CN'，代表中国大陆。

```
fake = Faker(locale='zh_CN')
print(fake.name())
# 刘婷
print(fake.address())
# 浙江省齐齐哈尔县南湖辛集街S座 525740
```

如果要生成中文繁体字，则可以传入代表中国台湾地区的locale值'zh\_TW'。

```
fake = Faker(locale='zh_TW')
print(fake.name())
# 馮雅琪
print(fake.address())
# 393 頭份縣天母巷90號2樓
```



## 7.3.4 Faker库---生成伪造数据

参数值	含义	参数值	含义
ar_EG	Arabic (Egypt)	hu_HU	Hungarian
ar_PS	Arabic (Palestine)	hy_AM	Armenian
ar_SA	Arabic (Saudi Arabia)	it_IT	Italian
bg_BG	Bulgarian	ja_JP	Japanese
bs_BA	Bosnian	ka_GE	Georgian (Georgia)
cs_CZ	Czech	ko_KR	Korean
de_DE	German	lt_LT	Lithuanian
dk_DK	Danish	lv_LV	Latvian
el_GR	Greek	ne_NP	Nepali
en_AU	English (Australia)	nl_NL	Dutch (Netherlands)
en_CA	English (Canada)	no_NO	Norwegian
en_GB	English (Great Britain)	pl_PL	Polish
en_NZ	English (New Zealand)	pt_BR	Portuguese (Brazil)
en_US	English (United States)	pt_PT	Portuguese (Portugal)
es_ES	Spanish (Spain)	ro_RO	Romanian
es_MX	Spanish (Mexico)	ru_RU	Russian
et_EE	Estonian	sl_SI	Slovene
fa_IR	Persian (Iran)	sv_SE	Swedish
fi_FI	Finnish	tr_TR	Turkish
fr_FR	French	uk_UA	Ukrainian
hi_IN	Hindi	zh_CN	Chinese (China Mainland)
hr_HR	Croatian	zh_TW	Chinese (China Taiwan)

### 7.3.4 Faker库---生成伪造数据

#### (3) 地址相关方法

`fake.address()` # 地址: '香港特别行政区大冶县上街钟街k座 664713'

`fake.building_number()` # 楼名: 'v座'

`fake.city()` # 完整城市名: '长春县'

`fake.city_name()` # 城市名字(不带市县): '梧州'

`fake.city_suffix()` # 城市后缀名: '市'

`fake.country()` # 国家名称: '厄立特里亚'

`fake.country_code(representation="alpha-2")` # 国家编号: 'BZ'

`fake.district()` # 地区: '沙湾'

`fake.postcode()` # 邮编: '332991'

`fake.province()` # 省: '河北省'

`fake.street_address()` # 街道地址: '武汉街D座'

`fake.street_name()` # 街道名称: '广州路'

`fake.street_suffix()` # 街道后缀名: '路'

### 7.3.4 Faker库---生成伪造数据

(3) 条形码、图书编号、身份证号码（社会安全码）、电话号码等相关

```
fake.ean(length=13) # EAN条形码: '5456457843465'
```

```
fake.isbn13(separator="-") # ISBN-13图书编号: '978-1-116-51399-8'
```

```
fake.ssn(min_age=18, max_age=90) # 身份证: '410622198603154708'
```

```
fake.phone_number() # 手机号码: '13334603608'
```

(4) 颜色相关

```
fake.hex_color() # 颜色十六进制值: '#a5cb7c'
```

```
fake.color_name() # 颜色名称: 'Orange'
```

```
fake.rgb_color() # 颜色RGB值: '15,245,42'
```

(5) 公司相关

```
fake.company() # 公司名: '鸿睿思博科技有限公司'
```

```
fake.job() # 工作职位: '游戏界面设计师'
```

### 7.3.4 Faker库---生成伪造数据

#### (6) 网络相关

```
fake.free_email() # email: 'jiejiao@hotmail.com'  
fake.ipv4_private() # 内部IP地址: '192.168.35.237'  
fake.ipv4_public() # 全球IP地址: 145.201.89.119  
fake.mac_address() # MAC地址: 59:e7:db:d3:91:d8  
fake.user_name() # 用户名: moxiulan  
fake.password() # 密码: 'KroNk0mC_2'
```

#### (7) 日期相关

```
fake.date(pattern="%Y-%m-%d", end_datetime=None) # 日期字符串  
# (可设置格式和最大日期): '1998-05-13'  
fake.date_time_between_dates(#日期字符串, 可设置起止日期  
    datetime_start=datetime(2000,12,31),  
    datetime_end=datetime(2003,12,31)).strftime('%Y-%m-%d') # '2002-02-16'  
fake.year() # 某年: '2016'  
fake.month() # 第几个月: '07'  
fake.day_of_month() # 几号: '23'  
fake.day_of_week() # 星期几: 'Tuesday'
```

## 第七章 模块

7.1 模块的概述

7.2 安装第三方模块

7.3 模块应用实例

7.4 在Python中调用R语言

7.5 实验

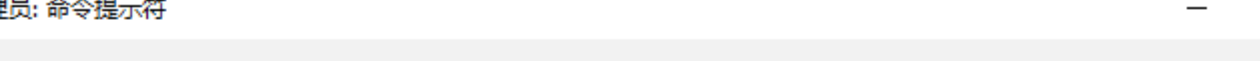
7.6 小结

7.7 习题

### 7.4.1 安装rpy2模块

## Python调用R的模块是rpy2。

## 安装rpy2, 命令行运行: C:\WINDOWS\system32>pip install rpy2



The screenshot shows a Windows command prompt window titled "选择管理员: 命令提示符". The command prompt shows the following commands and output:

```
C:\WINDOWS\system32>pip install rpy2
Collecting rpy2
  Using cached rpy2-3.3.4.tar.gz (174 kB)
Collecting pytest
  Downloading pytest-5.4.3-py3-none-any.whl (248 kB)
    | 225 kB 363 kB/s eta 0:00:01
```

安装成功后会提示如下：

```
Successfully built rpy2
Installing collected packages: pluggy, packaging, more-itertools, atomicwrites, wcwidth, py, col
orama, attrs, pytest, MarkupSafe, jinja2, pytz, tzlocal, pycparser, cffi, rpy2
Successfully installed MarkupSafe-1.1.1 atomicwrites-1.4.0 attrs-19.3.0 cffi-1.14.0 colorama-0.4
.3 jinja2-2.11.2 more-itertools-8.4.0 packaging-20.4 pluggy-0.13.1 py-1.9.0 pycparser-2.20 pytes
t-5.4.3 pytz-2020.1 rpy2-3.3.4 tzlocal-2.1 wcwidth-0.2.5

C:\WINDOWS\system32>
```

### 7.4.2 安装R语言工具

#### (1) 下载R语言工具

下载链接地址：<https://www.r-project.org/>。

点击R官网主页面上的 “download R”。

在跳转的镜像界面，下拉选择中国的镜像，我们选择第一个清华大学地址。

根据自己的操作系统选择相应的版本，我这里选择 Download R for Windows

【本机的操作系统是Windows 10】。

在base一行，点击 install R for the first time。

点击Download R 3.6.1 for Windows，保存到计算机。

#### (2) 安装

双击下载的可执行文件。

按照安装提示步骤一步步按照默认提示往下操作即可。

### 7.4.3 测试安装

(1)在Python Shell里面输入以下命令：

```
>>> import rpy2.objects as rob
```

没有任何错误提示即表示rpy2模块，R语言工具安装成功，可以进行R调用。



### 7.4.4 调用R示例

#### 1. python调用R对象

使用rpy2.objects包的r对象。

有三种语法格式调用R对象，分别“相当于”把r实例当作字典、把r实例当作方法、把r实例当作一个类对象。通过r实例，我们可以读取R的内置变量、调用R的函数、甚至直接把它当作R的解析器来用。

```
>>> import rpy2.objects as rob
```

**第一种**，把r实例当作字典：

```
>>> rob.r['pi']
```

```
R object with classes: ('numeric',) mapped to:
```

```
<FloatVector - Python:0x06ADE0A8 / R:0x08BBE9D8>
```

```
[3.141593]
```

### 7.4.4 调用R示例

**第二种**，把r实例当作方法：

```
>>> rob.r('pi')
```

```
R object with classes: ('numeric',) mapped to:
```

```
<FloatVector - Python:0x06ADCE68 / R:0x08BBE9D8>
```

```
[3.141593]
```

```
>>> rob.r('a<-c(1,2,3)')
```

```
R object with classes: ('numeric',) mapped to:
```

```
[1.000000, 2.000000, 3.000000]
```

**第三种**，把r实例当作一个类对象：

```
>>> rob.r.pi
```

```
R object with classes: ('numeric',) mapped to:
```

```
<FloatVector - Python:0x06ADE030 / R:0x08BBE9D8>
```

```
[3.141593]
```

这种方法从某种程度上讲是万能的，因为可以将任意大小和长度的R代码写成一个python字符串，之后通过rob.r('Rcode')调用执行。

### 7.4.4 调用R示例

`rob.r("r_script")` 可以执行R代码，比如 `t = rob.r('pi')` 就可以得到 R 中的PI（圆周率），返回的变量pi是一个向量，或者理解为python中的列表，通过 `t[0]` 就可以取出圆周率的值。

```
>>> t=rob.r('pi')
```

```
>>> t
```

```
R object with classes: ('numeric',) mapped to:
```

```
[3.141593]
```

```
>>> t[0]
```

```
3.141592653589793
```

```
>>> t=rob.r('a<-c(1,2,3)')
```

```
>>> t[0]
```

```
1.0
```

### 7.4.4 调用R示例

#### 2. 载入和使用R包

使用rpy2.objects.packages.importr对象，调用方法如下：

```
>>> from rpy2.objects.packages import importr
```

```
>>> base = importr('base')
```

```
>>> stats = importr('stats')
```

调用示例代码如下：

```
>>> stats.rnorm(10)
```

R object with classes: ('numeric',) mapped to:

```
<FloatVector - Python:0x06ADB5A8 / R:0x0A1E6F80>
```

```
[-0.921976, 0.049949, 0.306161, 1.092040, ..., -0.415047, 0.321349, -  
0.271509, 0.852308]
```

### 7.4.4 调用R示例

#### 3. 执行R的脚本文件

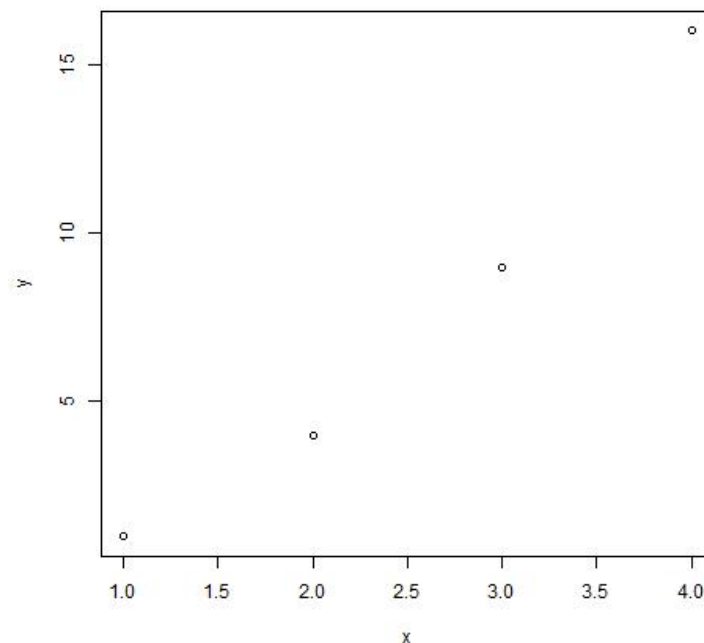
objects.r.source("file.r")可以执行r脚本文件。

```
import rpy2.objects as rob
```

```
rob.r.source('test01.r')  
x = rob.r('x') # 获取脚本里的变量  
y = rob.r('y')  
print(x) # [1] 1 2 3 4  
print(y) # [1] 1 4 9 16
```

test01.r的内容如下：

```
x <- c(1,2,3,4)  
y <- x*x  
jpeg(file="plot.jpg") # 保存图像  
plt <- plot(x,y) # 画散点图  
dev.off() # 关闭设备
```



## 第七章 模块

7.1 模块的概述

7.2 安装第三方模块

7.3 模块应用实例

7.4 在Python中调用R语言

7.5 实验

7.6 小结

7.7 习题

7.5.1 使用datetime模块

7.5.2 使用sys模块

7.5.3 使用与数学有关的模块

7.5.4 自定义和使用模块

## 第七章 模块

7.1 模块的概述

7.2 安装第三方模块

7.3 模块应用实例

7.4 在Python中调用R语言

7.5 实验

7.6 小结

7.7 习题



模块是一组Python代码的集合，可以使用其他模块，也可以被其他模块使用。

模块让你能够有逻辑地组织你的 Python 代码段。

模块能定义函数，类和变量，模块里也能包含可执行的代码。

如果要存储datetime，最佳方法是将其转换为timestamp再存储，因为timestamp的值与时区完全无关。

json序列化方法：dumps：无文件操作；dump：序列化+写入文件。

json反序列化方法：loads：无文件操作；load：读文件+反序列化。

## 第七章 模块

7.1 模块的概述

7.2 安装第三方模块

7.3 模块应用实例

7.4 在Python中调用R语言

7.5 实验

7.6 小结

7.7 习题

# 习题：

1. 在os.path模块中，用什么方法用来测试指定的路径是否为文件？
2. 在os模块中，用什么方法来返回包含指定文件夹中所有文件和子文件夹的列表？
3. 创建一个python包，在其中定义两个模块，一个模块中定义gcd(m,n), lcm(m,n), prime(n), fibonacci(n) 等函数,另一个模块中定义向量相加add(v1,v2)、向量相减subtract(v1,v2), 向量内积dot(v1,v2)、向量相乘multiply(v1,v2)、向量数乘scale(k,v)、向量相除divide(v1,v2)等函数。要求你的包支持模糊导入，并将包供他人使用测试。
4. 使用Faker库伪造一个个人信息（包括姓名、单位、职位、身份号码，电话号码、电子邮件、地址等）的字典，然后以JSON格式保存至文本文件。
5. 使用Faker库随机生成10个账号的用户名和初始密码，数据存储在列表中，元素是由用户名和初始密码构成的二元组。

感谢聆听

