Python语言

第七章　模块

7.1.1 模块与程序

我们写的代码保存的以.py结尾的Python文件就是一个独立的模块，模块包含了对象定义和语句。

如下所示代码：

```
#斐波那契数列（Fibonacci）
def Fib(n):
    '''
    斐波那契数列
    '''
    if (n<2):
        if (n == 0):
            return 0
        else :
            return 1
    else :
        return Fib(n-1)+Fib(n-2)
```

7.1.1 模块与程序

```
number = int(input('请输入一个正整数：'))
result = Fib (number)
print("%d 的斐波那契数列是：%d" % (number,result))
```

在上例中，我们定义了一个模块Fib，程序代码如上例所示。

上例代码运行结果如下：

```
请输入一个正整数：13
13 的斐波那契数列是：233
```

由此可见，**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。

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。

import module1, module2,…

导入模块后，就可以引用模块内的函数，语法格式如下：

模块名.函数名

```
import tkinter
# 实例化一个窗体对象
win = tkinter.Tk()
# 设置窗体的大小(300x300)，左上角(+150+150)
win.geometry("300x100+150+150")
win.mainloop()
```

7.1.3 模块导入方法

注意事项：

(1)在VSCode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：（仅从模块中导入指定的函数)

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。

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。

```
import tkinter as tk

# 实例化一个窗体对象
win = tk.Tk()
# 设置窗体的大小(300x300)，左上角(+150+150)
win.geometry("300x100+150+150")

win.mainloop()
```

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：

在开发目录下，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VSCode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（或下级目录），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

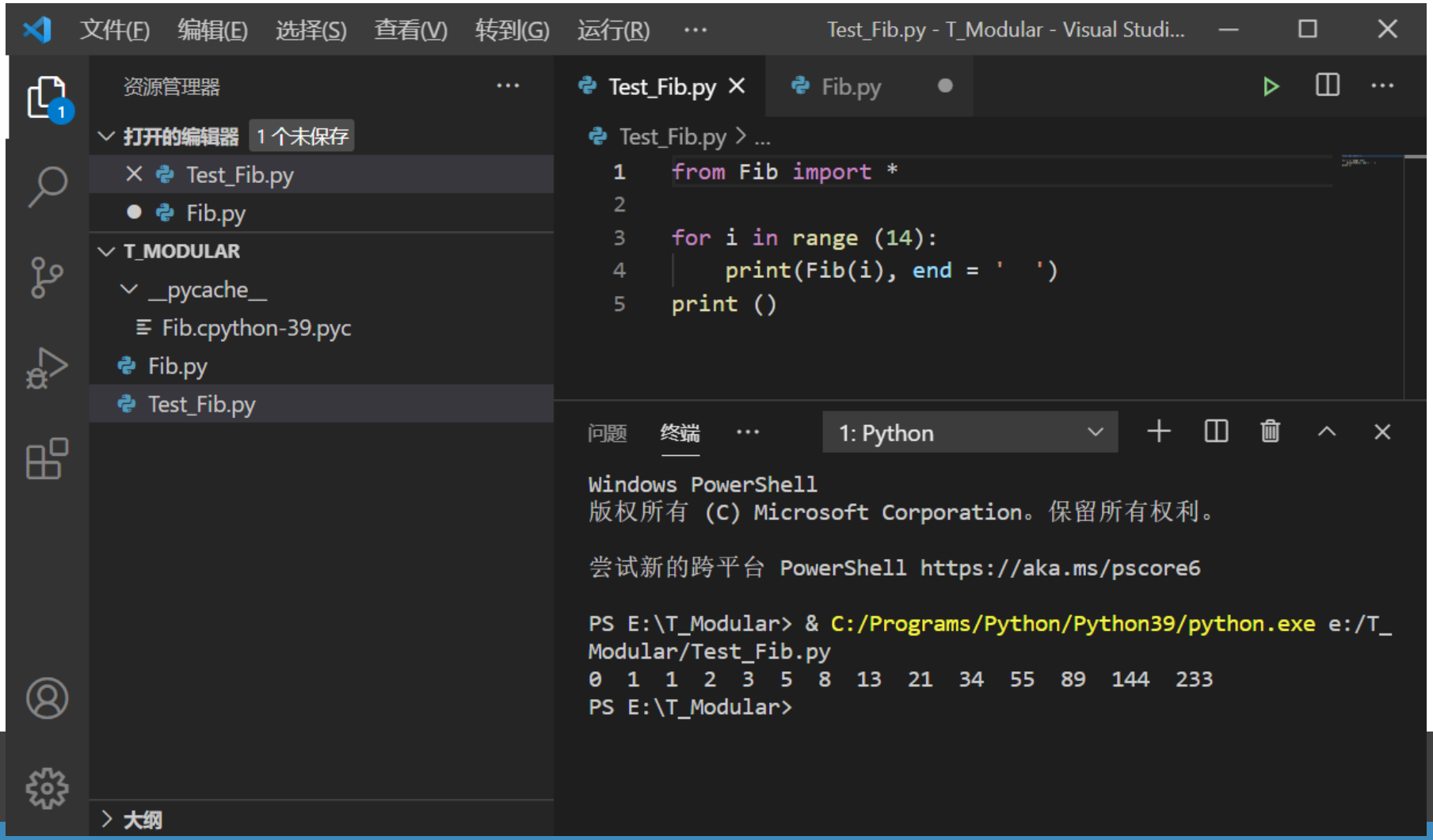
7.1.4 自定义模块和包

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。

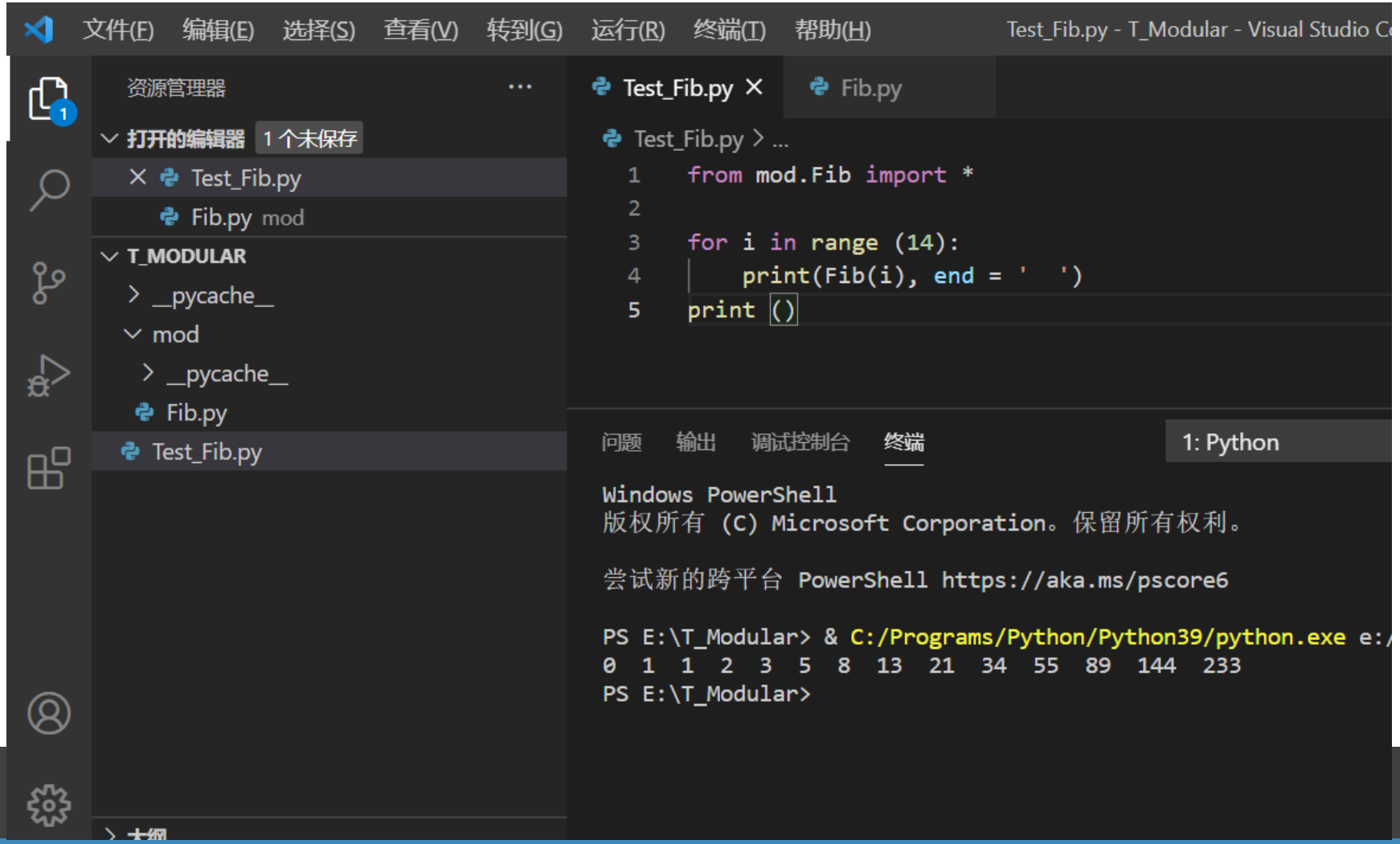
7.1.4 自定义模块和包

自定义模块 在当前文件所在目录下



7.1.4 自定义模块和包

自定义模块 在当前文件所在目录 的子目录下



第七章　模块

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 3.6.5安装为例，确保安装时勾选选了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 3.6.5

(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 10.0.1 from c:\users\administrator\appdata\local\programs\python\
python36-32\lib\site-packages\pip (python 3.6
```

(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：

```
Successfully installed pip-10.0.1 setuptools-39.2.0 wheel-0.31.1
```

(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。

本节我们仅以从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。

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：

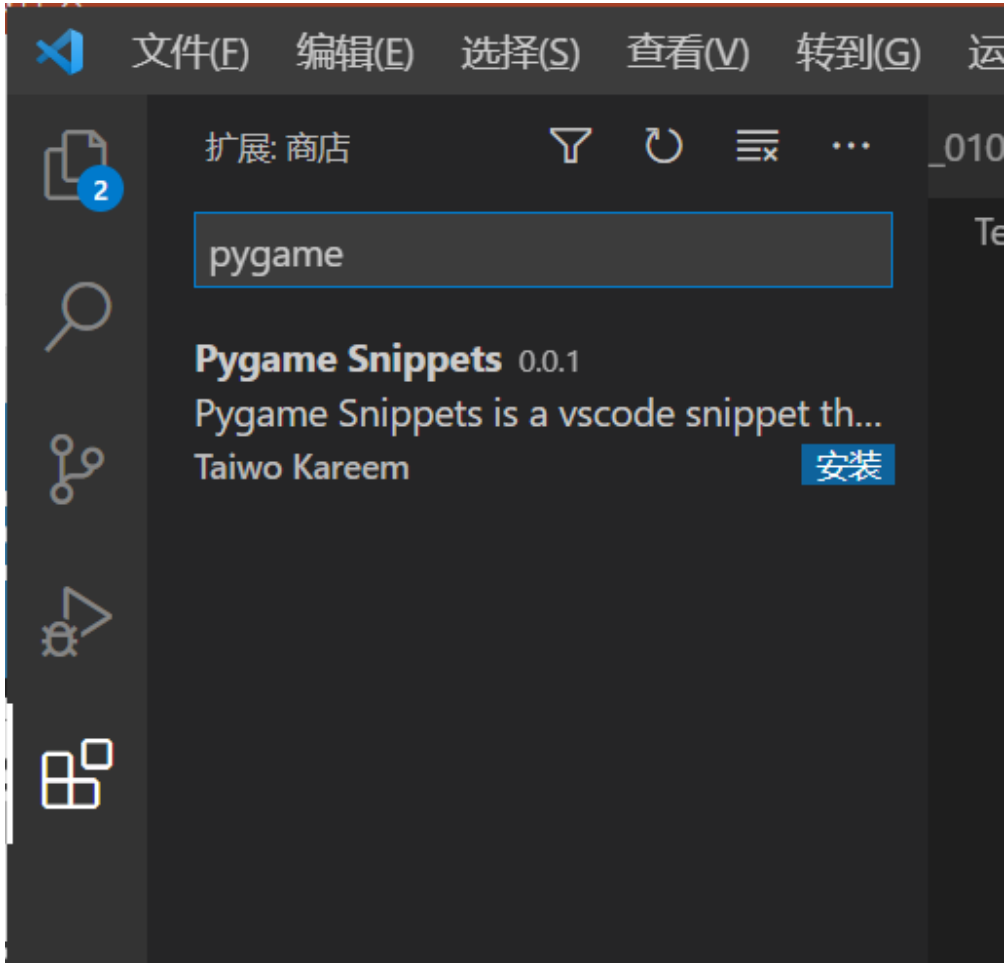
```
>>> import web3
```

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。

VSCode 下在线安装



第七章　模块

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
```

运行结果如下：

```
>>> print(now)
2018-06-19 13:07:58.726038
```

7.3.1 日期时间相关：datetime模块

```
>>> print(type(now))
<class 'datetime.datetime'>
```

从上例可以看出，datetime是模块，datetime模块还包含一个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。

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(2018, 6, 19, 13, 15) # 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
1529385300.0
```

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fromtime

stamp()方法，如下所示代码：

```
>>> from datetime import datetime
>>> t = 1529385300.0
```

运行结果如下：

```
>>> print(datetime.fromtimestamp(t))
2018-06-19 13:15:00
```

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 = 1529385300.0
```

7.3.1 日期时间相关：datetime模块

运行结果如下：

```
>>> print(datetime.fromtimestamp(t)) # 本地时间
2018-06-19 13:15:00
>>> print(datetime.utcfromtimestamp(t)) # UTC时间
2018-06-19 05:15:00
```

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('2018-06-19 13:15:00', '%Y-%m-%d %H:%M:%S')
```

7.3.1 日期时间相关：datetime模块

运行结果如下：

```
>>> print(datee_test)

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

在上例中，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()
```

运行结果如下：

```
>>> print(now.strftime('%a, %b %d %H:%M'))
Tue, Jun 19 13:07
```

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，如下例所示代码：

```
>>> from datetime import datetime, timedelta
>>> now = datetime.now()
```

运行结果如下：

```
>>> now
```

7.3.1 日期时间相关：datetime模块

datetime.datetime(2018, 6, 19, 14, 42, 36, 664596)

>>> now + timedelta(hours=10)

datetime.datetime(2018, 6, 20, 0, 42, 36, 664596)

>>> now - timedelta(days=10)

datetime.datetime(2018, 6, 9, 14, 42, 36, 664596)

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

datetime.datetime(2018, 7, 2, 13, 42, 36, 664596)

从上例可见，使用timedelta可以很容易地算出前几天和后几天的时刻。

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

运行结果如下：

```
>>> now

datetime.datetime(2018, 6, 19, 15, 20, 8, 373839)

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

>>> dt_test

datetime.datetime(2018, 6, 19, 15, 20, 8, 373839,
tzinfo=datetime.timezone(datetime.timedelta(0, 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)
```

运行结果如下：

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

```
2018-06-19 07:27:27.085313+00:00
```

7.3.1 日期时间相关：datetime模块

将转换时区为北京时间，如下所示代码：

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

运行结果如下：

```
>>> print(bj_dtime)
2018-06-19 15:27:27.085313+08:00
```

从上例可见，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，里面可以包含方括号括起来的数组，即python里面的列表。

在python中，json模块专门处理json格式的数据，提供了四种方法： dumps、dump、loads、load。

(1) dumps、dump：

dumps、dump实现序列化功能，但在使用功能上有差别。其中，dumps实现的是将数据序列化为字符串(str)，而在使用dump时，必须传文件描述符，将序列化的字符串(str)保存到文件中。

7.3.2 读写JSON数据：json模块

Dumps方法的使用，如下所示：如下所示代码：

```
>>> import json
```

运行结果如下：

```
>>> json.dumps('Python') #字符串
'"Python"'
>>> json.dumps(12.78) #数字
'12.78'
>>>dict_test = {"teacher_name":"Mr.Liu","teach_age":18}#字典
```

7.3.2 读写JSON数据：json模块

```
>>> json.dumps(dict_test) #字典
'{"teacher_name": "Mr.Liu", "teach_age": 18}'
```

在上例中，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。

Dump方法的使用，如下所示：如下所示代码：

```
import json
dict_test = {"teacher_name":"Mr.Liu","teach_age":18}
with open("G:\\json_test.json","w",encoding='utf-8') as file_test:
    json.dump(dict_test,file_test,indent=4)
```

7.3.2 读写JSON数据：json模块

在上例中，使用dump方法将字典数据dict_test保存到G盘根目录下的json_test.json文件中。运行后的效果如下所示：

```
{
    "teacher_name": "Mr.Liu",
    "teach_age": 18
}
```

7.3.2 读写JSON数据：json模块

(2) loads、load：

Loads、load是反序列化方法。loads 只完成了反序列化，load 只接收文件描述符，完成了读取文件和反序列化。

Loads方法的使用，如下所示代码：

```
>>> import json
>>> json.loads('{"teacher_name":"Mr.Liu","teach_age":18}')
```

运行结果如下：

```
{'teacher_name': 'Mr.Liu', 'teach_age': 18}
```

7.3.2 读写JSON数据：json模块

在上例中，loads将已经序列化的字典字符串数据反序列化为字典数据。

Load方法的使用，如下所示代码：

```
import json
with open("G:\\json_test.json", "r", encoding='utf-8') as file_test:
    test_loads = json.loads(file_test.read())
    file_test.seek(0)
    test_load = json.load(file_test) # 同 json.loads(file_test.read())
print(test_loads)
```

7.3.2 读写JSON数据：json模块

print(test_load)

在上例中，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。运行结果如下所示：

{'teacher_name': 'Mr.Liu', 'teach_age': 18}

{'teacher_name': 'Mr.Liu', 'teach_age': 18}

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)
```

运行结果如下：

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

7.3.3 系统相关：sys模块

以上命令，显示了sys模块可用的方法。

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。

运行命令，如下所示代码：

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

运行结果如下：

```
['G:/Python教材编写/例子20180619/例子',
'C:\\Users\\Lenovo\\AppData\\Local\\Programs\\Python\\Python36-32\\Lib\\idlelib',
```

7.3.3 系统相关：sys模块

'C:\\Users\\Lenovo\\AppData\\Local\\Programs\\Python\\Python36-32\\python36.zip',
'C:\\Users\\Lenovo\\AppData\\Local\\Programs\\Python\\Python36-32\\DLLs',
'C:\\Users\\Lenovo\\AppData\\Local\\Programs\\Python\\Python36-32\\lib',
'C:\\Users\\Lenovo\\AppData\\Local\\Programs\\Python\\Python36-32',
'C:\\Users\\Lenovo\\AppData\\Local\\Programs\\Python\\Python36-32\\lib\\site-packages']

从上面的运行结果可以看出，该命令获取了指定模块搜索路径的字符串集合。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。

运行该命令，如下所示代码：

```
>>> sys.argv
```

运行结果如下：

```
['G:/Python教材编写/例子20180619/例子/json_load_test.py']
```

从上面的运行结果可以看出，sys.argv 变量是一个包含了命令行参数的字符串列表,利用命令行向程序传递参数。其中,脚本的名称是 sys.argv 列表的第一个参数。

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。通过运行以下命令，导入该模块，如下所示代码：

```
>>> import math
```

通过dir(math)命令查看math模块可用的方法,例如：

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

运行结果如下：

```
['__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',
```

7.3.4 数学:math模块

'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。通过运行以下命令，导入该模块：

```
>>> import random
```

通过dir(random)命令查看random模块可用的方法,运行如下命令：

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

运行结果如下：

['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

运行以下代码，得到如下结果：

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

7.3.5 随机数:random模块

(4)洗牌：shuffle()

运行以下代码：

```
>>> test_shuffle = ['A','Q',1,6,7,9]

>>> random.shuffle(test_shuffle)
```

运行结果如下：

```
>>> test_shuffle

[7, 6, 1, 'A', 'Q', 9]
```

第七章　模块

R是用于统计分析、绘图的语言和操作环境。R是属于GNU系统的一个自由、免费、源代码开放的软件，它是一个用于统计计算和统计制图的优秀工具。

要在Python中调用R语言的前提条件是：要在本机安装rpy2模块和R语言工具。

7.4.1 安装rpy2模块

Python调用R的模块是rpy2。

安装：注意 rpy2 的版本和 python 的版本要对应，也要和 R 的版本对应。

【以下操作以本Python版本为例：Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32】

(1)下载rpy2。

下载链接地址：http://www.lfd.uci.edu/~gohlke/pythonlibs/#rpy2。

选择下载 rpy2-2.9.4-cp36-cp36m-win32.whl，对应python3.6.5且32位版本。

(2)安装。

7.4.1 安装rpy2模块

在命令提示符下，输入以下命令：

pip install G:\Python教材编写\安装文件\RPY2\ rpy2-2.9.4-cp36-cp36m-win32.whl

安装成功后会提示如下：

Successfully built MarkupSafe

Installing collected packages: MarkupSafe, jinja2, rpy2

Successfully installed MarkupSafe-1.0 jinja2-2.10 rpy2-2.9.4

至此，rpy2就成功安装到计算机中了。

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 3.5.0 for Windows，保存到计算机。

7.4.2 安装R语言工具

(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.robjects 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.robjects包的r对象。调用方法如下：

```
>>> import rpy2.robjects as rob
```

有三种语法格式调用R对象：

第一种，实验代码如下：

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

实验代码运行效果如下：

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

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

7.4.4 调用R示例

[3.141593]

第二种，实验代码如下：

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

实验代码运行效果如下：

```
R object with classes: ('numeric',) mapped to:
<FloatVector - Python:0x06ADCE68 / R:0x08BBE9D8>
[3.141593]
```

第三种，实验代码如下：

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

7.4.4 调用R示例

实验代码运行效果如下：

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

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

[3.141593]

7.4.4 调用R示例

2. 载入和使用R包

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

```
>>> from rpy2.robjects.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>
```

7.4.4 调用R示例

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

第七章　模块

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

7.6 小结

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

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？

感谢聆听