PYTHON 语言

岭南师范学院

第9周 Python模块

教学目标

- 1 理解模块与包的概念及用法
- 2 掌握Python内置模块的基本使用方法
- 3 掌握第三方库的使用

主要内容

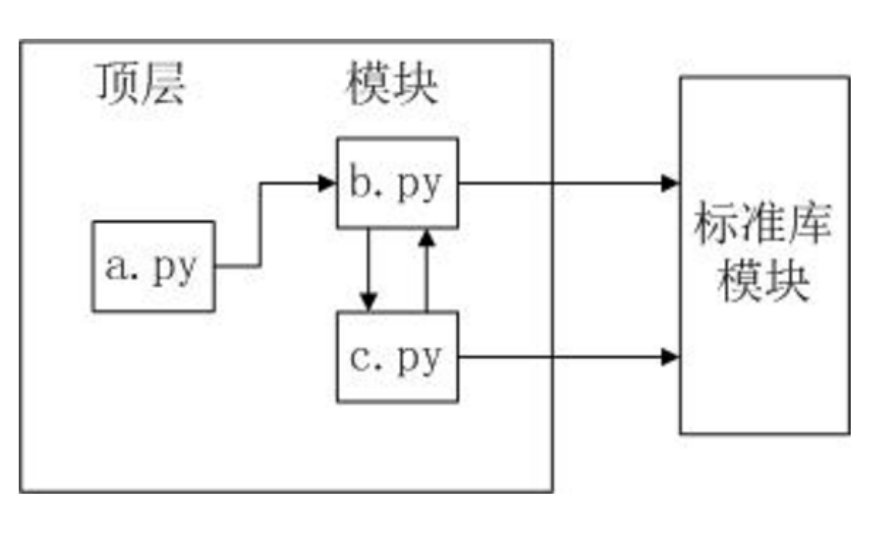
- 7.1 模块的定义
- 7.2 模块的使用
- 7.3 包的使用
- 7.4 标准库
- 7.5 自定义库
- 7.6 第三方库

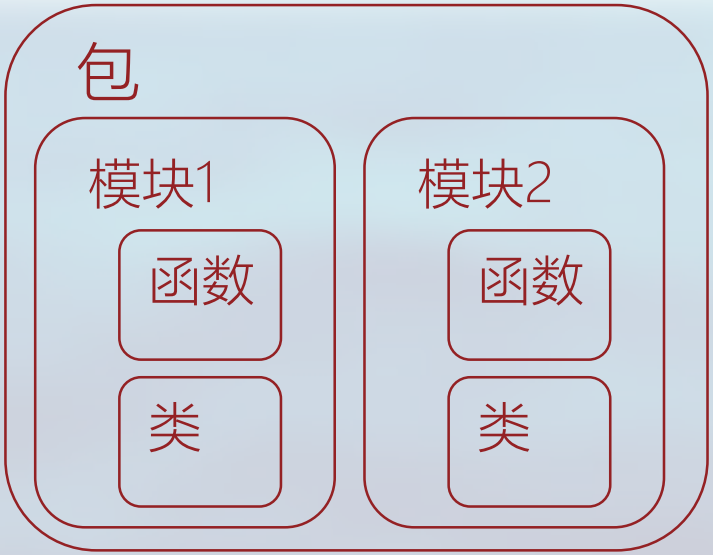
7.1 Python程序的架构

- Python程序是由包、模块、函数组成的。其中，包是由一系列模块组成的集合，而模块是处理某一类问题的函数或（和）类的集合。
- 在Python中，模块就是一个包含变量、函数或类的定义的程序文件，除了各种定义之外，还可包含其他的各种Python语句。
- 在大型系统中，往往将系统功能分别使用多个模块来实现或者将常用功能集中在一个或多个模块文件中，然后在顶层的主模块文件或其它文件中导入使用。
- Python本身也提供了大量内置模块，并可集成各种扩展模块。

7.1 Python程序的架构

- Python源代码文件：*.py
  - 一个py文件是一个模块（module）
  - 多个模块可以组成一个包（package）





7.1.1 模块的作用

- 模块是Python中的最高级别组织单元，它将程序代码和数据封装起来以便重用
- 模块的三个角色：
  1. 代码重用
  2. 系统命名空间的划分（模块可理解为变量名的封装，即模块就是命名空间）
  3. 实现共享服务和数据

7.2 导入模块

- 应用程序要使用模块中的变量或函数，需要先导入该模块
  - 导入从本质上讲，就是在一个文件中载入另一个文件，并且能够读取那个文件的内容。一个模块内的内容通过这样的方法其属性（object，attribute）能够被外界使用。

- 导入的方式
  - **import x**
  - **from x import ***
  - **from x import a, b, c**
  - **x = __import__('x')**

7.2.2 导入方式

- 1.import x语句, 用于导入整个模块
  - import module1, module2.... # 建议一个import语句只导入一个模块

    import module as module_alias # 别名（也就是自定义的模块名称空间）
  - 导入模块x，并在当前命名空间（namespace）创建该模块的引用。可以使用：x.name引用定义在模块X中的属性。

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.sin(math.pi/2)
1.0
>>> math.tan(math.pi/4)
0.9999999999999999
```

- 2. from x import * ：
  - 导入模块x，并在当前命名空间，创建该模块中所有公共对象（名字不以__开头）的引用。即你能使用普通名字（直接是name）去引用模块X中的属性，但是x本身没有定义，不能使用x.name。并且如果命名空间中原来有同名的name定义时，它将会被新的name取代
- 3 from x import a, b, c：
  - 导入模块x，并在当前命名空间创建该模块给定对象的引用。

```
>>> from math import *
>>> pi
3.141592653589793
>>> ceil(4.567)
5
```

```
>>> from math import sin,cos
>>> sin(40)
0.745113160479348 8
>>> pi
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    pi
NameError: name 'pi' is not defined
```

- 注意：
  - 在使用import导入模块时,模块中的变量名使用“模块名.”作为限定词，所以不存在歧义，即使与其它模块变量同名也没有关系。在使用from时，当前模块的同名变量引用了模块内部的对象。在遇到与当前模块或其它模块变量同名时，使用时应特别注意。

7.2.3 执行语句

- import和from语句在执行导入操作时，会执行被导入的模块。模块中的赋值语句执行时创建变量，def语句执行时创建函数对象。总之，模块中的全部语句都会被执行，且只执行一次。当再次使用import或from语句导入模块时，不会执行模块代码，只是重新建立到已经创建的对象的引用而已。所以，import和from语句是隐性的赋值语句。
  - Python执行import语句时，创建一个模块对象和一个与模块文件同名的变量，并建立变量和模块对象的引用。
  - Python执行from语句时，会同时在当前模块和导入模块中创建同名变量，并引用模块在执行时创建的对象。

7.2.4 重新加载模块（reload内置函数）

- 很多时候，再次使用import和from导入模块时，其本意通常是重新执行模块代码，恢复相关变量到模块执行时的状态。显然，这种愿望通过再次使用import和from导入是无法达到的。
- 因此，Python在imp模块中提供了reload函数来重新载入并执行模块代码。使用reload重载模块时，如果模块文件已经被修改，则会执行修改后的代码。
- reload函数用模块变量名作为参数，重载对应模块，所以reload重载的必须是使用import语句已经导入的模块。

- 例：

```
>>>from imp import reload
>>> reload(os)
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    reload(os)
NameError: name 'os' is not defined
>>> import os
>>> reload(os)
<module 'os' from
'C:\\Users\\shifeng\\AppData\\Local\\Programs\\Python\\Python37\\lib\\os.py'
```

7.2.5 模块搜索路径

1. 找到模块文件：在模块搜索路径下搜索模块文件

   程序的主目录
   PYTHONPATH目录
   标准链接库目录

- 2.编译成字节码：文件导入时会编译，因此，顶层文件的.pyc字节码文件在内部使用后会被丢弃，只有被导入的文件才会留下.pyc文件
- 3.执行模块的代码来创建其所定义的对象：模块文件中的所有语句从头至尾依次执行，而此步骤中任何对变量名的赋值运算，都会产生所得到的模块文件的属性

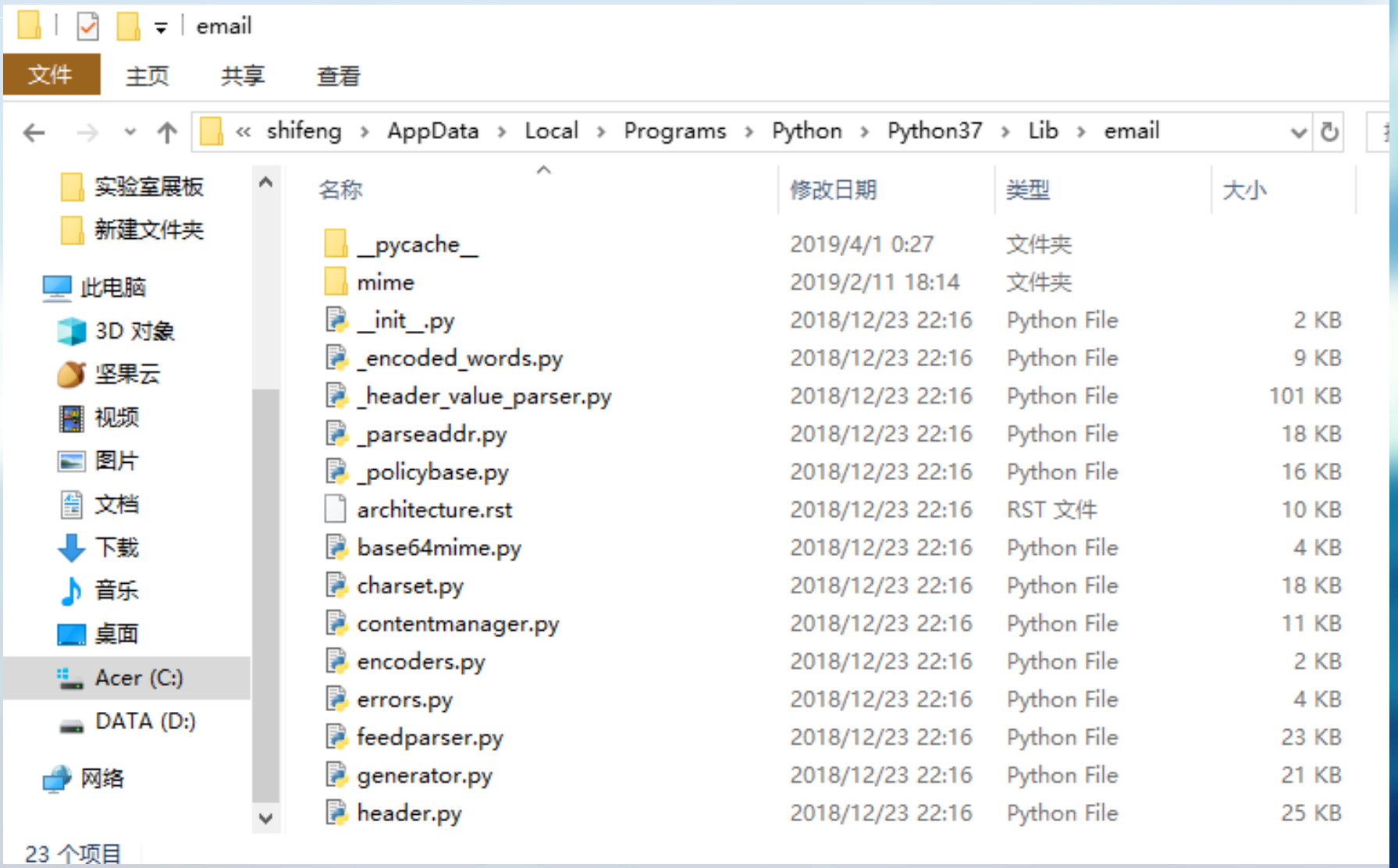
注意

- 模块只在第一次导入时才会执行如上步骤，后续的导入操作只不过是提取内存中已加载的模块对象，reload()可用于重新加载模块。
  - 模块并不是用来执行操作（如打印文本）的，而是用于定义变量、函数、类等。
  - 定义只需做一次，因此导入模块一次和多次效果相同。
  - 导入一次是重要的优化，可以防止无穷的导入循环。
- 导入模块时，需要能查找到模块的文件路径。
- 导入模块时，不能在 import或from语句中指定模块文件的路径，只能使用Python设置的搜索路径。

7.3 包

- 在大型系统中，通常会根据代码功能将模块文件放在多个目录中。在导入这种位于目录中的模块文件时，需要指定目录路径。Python将存放模块文件的目录称为包。包是一个有层次的文件目录结构，它定义了一个由模块和子包组成的Python应用程序执行环境。包可以解决如下问题：
  - 把命名空间组织成由层次的结构；
  - 允许程序员把有联系的模块组合到一起；
  - 允许程序员使用有目录结构而不是一大堆杂乱无章的文件；
  - 解决有冲突的模块名称

- 功能相似的模块使用包组成层次组织结构



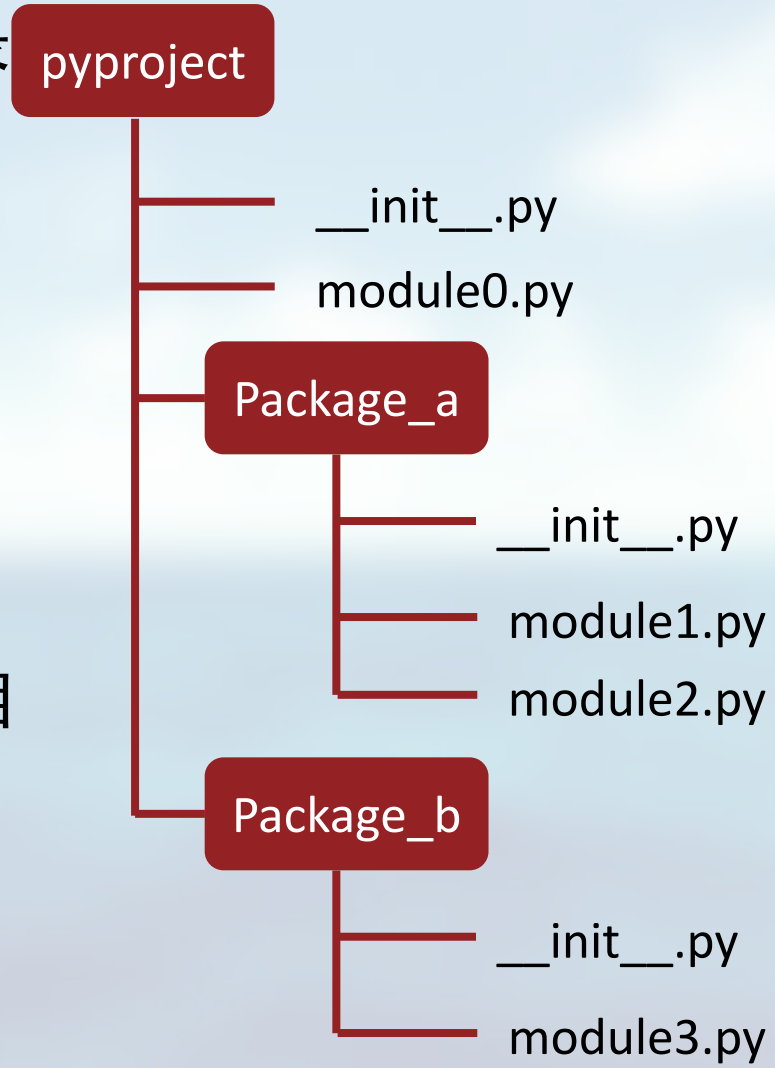
7.3.1 包的导入与使用

- import [包名1.[包名2….]].模块名　　#导入包中模块
- from [包名1.[包名2….]].模块名 import 成员名 #导入模块中的具体成员
- from 包名 import *

  from [包名1.[包名2….]].模块名.函数名　#使用全限制调用模块的成员

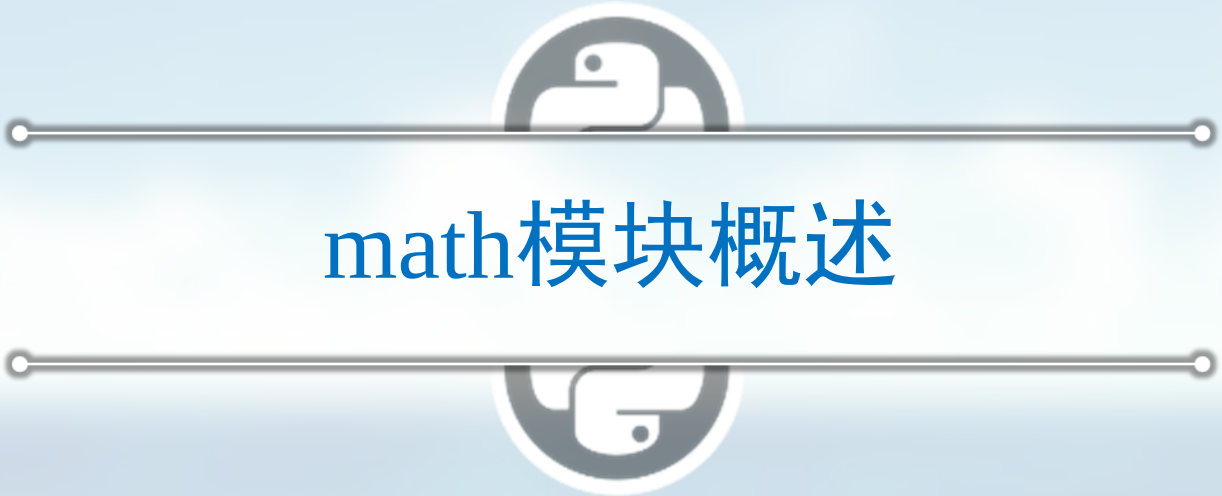
7.3.2 创建包

- 在指定目录中创建对应包名的目录
- 在该目录下创建一个特殊文件：__init__.py文件
- 最后在该目录下创建模块文件
- 【例】创建包示例。在D:\python\w7\目录中，创建如下目录结构



7.4 Python标准库

- Python标准库，也称内置库或内置模块，是Python的组成部分，它随Python解释器一起安装在系统中。
- 7.4.1 math库
- 7.4.2 random模块
- 7.4.3 time模块



math模块概述

math库

- math库是Python内置的数学函数库，提供支持整数和浮点数运算的函数。
- math库共提供了4个数学常数和44个函数，分为数值运算函数、幂对数函数、三角对数函数和高等特殊函数等类型。
- 执行dir(math)函数命令，

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```

random模块概述

random模块

- random库中的函数主要用于产生各种分布的伪随机数序列。
- 概率是确定、可见的，被称为伪随机数。
- 可以生成不同类型的随机数函数，最基本是 **random.random()**，它生成一个[0.0, 1.0)之间的随机小数，其它函数扩展其实现。
- https://docs.python.org/zh-cn/3.7/library/random.html

```
>>>from random import *
>>>random()
0.5780913011344704
>>>random()
0.20609823213950174
```

random库的常用函数

函数	描述
seed(a=None)	初始化随机数种子，默认值为当前系统时间
random()	生成一个[0.0, 1.0)之间的随机小数
randint(a, b)	生成一个[a,b]之间的整数
getrandbits(k)	生成一个k比特长度的随机整数
randrange(start, stop[, step])	生成一个[start, stop)之间以step为步数的随机整数
uniform(a, b)	生成一个[a, b]之间的随机小数
choice(seq)	从序列类型(例如：列表)中随机返回一个元素
shuffle(seq)	将序列类型中元素随机排列，返回打乱后的序列
sample(pop, k)	从pop类型中随机选取k个元素，以列表类型返回

random库与随机数运用

- random库使用random.seed(a)对后续产生的随机数设置种子a。

```
>>>from random import *
>>>seed(10)
>>>random()
0.5714025946899135
>>>random()
0.4288890546751146
>>>seed(10)     #再次设置相同的种子，则后续产生的随机数相同
>>>random()
0.5714025946899135
>>>random()
0.4288890546751146
```

■设置随机数种子的好处是可以**准确复现随机数序列**，用于重复程序的运行轨迹。对于仅使用随机数但不需要复现的情形，可以不用设置随机数种子。

■如果程序没有显式设置随机数种子，则使用随机数生成函数前，将默认以当前系统的运行时间为种子产生随机序列。

```
import random
print(random.randint(0,100))                    #随机生成一个0~100的整数
print(random.randrange(0,101,2))                #随机生成一个0~100的偶数
print(random.random())                          #随机生成一个0~1间的浮点数
print(random.choice('jklhgy&#@’)) #从指定字符集合里随机获取一个字符
list = [1, 2, 3, 4, 5, 6]
random.shuffle(list)                            #将一个列表中的元素打乱
print(list)
[6, 2, 5, 1, 3, 4]
print(random.sample(list,3))          #从指定序列中随机获取指定长度的片段
[6, 3, 1]
```

time模块概述

time库概述

- 处理时间是程序最常用的功能之一，time库是Python提供的处理时间标准库。time库提供系统级精确计时器的计时功能，可以用来分析程序性能，也可让程序暂停运行时间。

- time库的功能主要分为3个方面：**时间处理、时间格式化和计时**。
  - 时间处理主要包括4个函数：time.time()、time.gmtime()、time.localtime()、time.ctime()。
  - 时间格式化主要包括3个函数：time.mktime()、time.strftime()、time.strptime()。
  - 计时主要包括3个函数：time.sleep()、time.monotonic()、time.perf_counter()

时间处理

- 计算机可以使用时间戳、格式化时间的字符串和struct_time元组三种方式表示时间。
  - Unix时间戳(Unix timestamp)，或称Unix时间(Unix time)、POSIX时间(POSIX time)，是一种时间表示方式，定义为从格林威治时间1970年01月01日00时00分00秒(北京时间1970年01月01日08时00分00秒)起至现在的总秒数。Unix时间戳不仅被使用在Unix系统、类Unix系统中(比如Linux系统)，也在许多其他操作系统中被广泛采用。

■使用time.time()获取当前时间戳

```
>>>import time
>>>time.time()
1554830125.1918387
```

- 使用time.gmtime(secs)获取当前时间戳对应的struct_time对象

```
>>> time.gmtime(time.time())
time.struct_time(tm_year=2019, tm_mon=4,
tm_mday=9, tm_hour=17, tm_min=10, tm_sec=28,
tm_wday=1, tm_yday=99, tm_isdst=0)
```

- 使用time.localtime(secs)获取当前时间戳对应的本地时间的struct_time对象

```
>>> time.localtime(now)
time.struct_time(tm_year=2019, tm_mon=4,
tm_mday=10, tm_hour=1, tm_min=10, tm_sec=48,
tm_wday=2, tm_yday=100, tm_isdst=0)
```

- 注意结果与gmtime的区别，UTC时间已自动转换为北京时间。

struct_time元组包含9个元素：

- year，4位的年份，例如2019
- month，月份，1~12的整数
- day，日期，1~31的整数
- hours，小时，0~23的整数
- minutes，分钟，0~59的整数
- seconds，秒钟，0~59的整数
- weekday，星期，0~6的整数，星期一为0
- Julian day，一年有几天，1~366的整数
- DST，表示是否为夏令时。如果DST等于0，则给定的时间属于标准时区；如果DST等于1，则给定的时间属于夏令时时区

- 使用time.ctime(secs)获取当前时间戳对应的易读字符串表示，内部会调用time.localtime()函数以输出当地时间。

```
>>> time.ctime(now)
'Wed Apr 10 01:18:10 2019'
```

时间格式化

- 使用time.mktime(t) 将struct_time对象t转换为时间戳，注意t代表当地时间。

```
>>> t=time.localtime(time.time())
>>> time.mktime(t)
1554830483.0
>>> time.ctime(time.mktime(t))
'Wed Apr 10 01:21:23 2019'
```

- time.strftime()函数是时间格式化最有效的方法，几乎可以以任何通用格式输出时间。该方法利用一个格式字符串，对时间格式进行表达。

```
>>> lctime = time.localtime()
>>> lctime
time.struct_time(tm_year=2019, tm_mon=4, tm_mday=10, tm_hour=1,
tm_min=23, tm_sec=51, tm_wday=2, tm_yday=100, tm_isdst=0)
>>> time.strftime("%Y-%m-%d %H:%M:%S", lctime)
'2019-04-10 01:23:51'
```

格式字符串中可以使用的日期和时间符号如下：

%y 两位数的年份表示（00-99）

%Y 四位数的年份表示（0000-9999）

%m 月份（01-12）

%d 月内中的一天（01-31）

%H 24小时制小时数（0-23）

%I 12小时制小时数（01-12）

%M 分钟数（00-59）

%S 秒（00-59）

%a 本地简化星期名称

%A 本地完整星期名称

%b 本地简化的月份名称

%B 本地完整的月份名称

%c 本地相应的日期表示和时间表示

%j 年内的一天（001-366）

%p 本地A.M.或P.M.

%U 一年中的星期数（00-53），星期天为星期的开始

%w 星期（0-6），星期天为星期的开始

%W 一年中的星期数（00-53），星期一为星期的开始

%x 本地相应的日期表示

%X 本地相应的时间表示

%Z 当前时区的名称

%% %号本身

- strptime()方法与strftime()方法完全相反，用于提取字符串中时间来生成strut_time对象，可以很灵活的作为time模块的输入接口

```
>>> timeString = '2019-04-26 12:55:20'
>>> time.strptime(timeString, "%Y-%m-%d %H:%M:%S")
time.struct_time(tm_year=2019,    tm_mon=4,    tm_mday=26,
tm_hour=12,  tm_min=55,  tm_sec=20,  tm_wday=4,  tm_yday=26,
tm_isdst=-1)
```

- 更多功能

  https://docs.python.org/zh-cn/3.7/library/time.html

7.5 创建模块

- 创建模块，即创建一个.py文件，在其中包含用于完成任务的变量、类和函数，不包括main函数。

- 模块使用之前要导入该模块，导入方法之前已做过介绍。

- 例7.1 在模块中定义了算术四则运算

```
PI = 3.14                #定义常量
def add(x, y):           #定义函数
    return x + y         #加
def sub(x, y):           #定义函数
    return x - y         #减
def mul(x, y):           #定义函数
    return x * y         #乘
def div(x, y):           #定义函数
    return x / y         #除
```

将此代码保存到某个路径下，如 D:\Python\MyMath.py

- 例7.2 自定义模块调用

  假定【例7.1】中创建的模块mymath.py保存在与例7.2.py同一目录下。
  或者在IDLE交互界面上输出如下代码

```
>>> import sys
>>> sys.path.append('D:\python') #临时增加模块搜索路径
```

调用方式一：

```
>>> import mymath
>>> mymath.PI
3.14
>>> mymath.add(3,4)
7
>>> mymath.div(3,4)
0.75
>>> mymath.sub(5,3)
2
>>> mymath.mul(3,4)
12
```

调用方式二：

```
>>> from mymath import *
>>> PI
3.14
>>> add(3,4)
7
>>> sub(3,4)
-1
>>> mul(3,4)
12
>>> div(3,4)
0.75
```

python模块中的__name__属性

- python 里__name__属性是一个特殊变量：
- 1、__name__是一个变量。前后加了双下划线是因为是因为这是系统定义的名字。普通变量不要使用此方式命名变量。
- 2、Python有很多模块，而这些模块是可以独立运行的！这点不像C++和C的头文件。
- 3、import的时候是要执行所import的模块的。

- 4、__name__就是标识模块的名字的一个系统变量。这里分两种情况：
  - 假如当前模块是主模块（也就是调用其他模块的模块），那么此模块名字就是__main__，通过if判断这样就可以执行“__mian__:”后面的主函数内容；
  - 假如此模块是被import的，则此模块名字为文件名字（不加后面的.py），通过if判断这样就会跳过“__mian__:”后面的内容。
- 通过上面方式，python就可以分清楚哪些是主函数，进入主函数执行；并且可以调用其他模块的各个函数等等。

- 定义一个hello.py模块如下

```
def hello():
    print("调用hello函数")


if __name__ == '__main__':
    hello();
```

由于此程序是主程序，所以一旦执行因为__name__ == '__main__'，进入if，所以会执行hello函数。

```
import hello

print(hello.__name__)
```

现在hello模块中的__name__=='hello'，不再是'__main__'了。
所以hello模块中 不在进入if语句。

7.6 第三方库

- 7.6.1 第三方库介绍
- 7.6.2 安装第三方库
- 7.6.3 pyinstaller
- 7.6.4 jieba库
- 7.6.5 wordcloud 库

7.6.1 第三方库介绍

- Python官方网站提供了第三方库索引功能（the Python Package PyPI），网址如下。https://pypi.org
- 该站点列出Python语言超过14万个第三方库的基本信息。
- 当前流行的编程思想是“模块编程”。
- 用户程序包括标准库、第三方库、用户程序、程序运行所需要的资源等，用搭积木类似的方法组建程序，这就是模块编程。
- “模块编程”思想强调充分利用第三方库，编写程序的起点不再是探究每个程序算法或功能的设计，而是尽可能探究运用库函数编程的方法。

7.6.2 安装

- 第三方库需要安装后才能使用。用户下载第三方库后，可以根据软件文档来安装或使用pip工具安装。
- pip工具由Python官方提供并维护，是常用且高效的在线第三方库安装工具。
- pip3是 Python的内置命令，用于 Python3版本安装第三方库，需要在命令行下执行。

（1）pip3 –help

（2）pip3 install <拟安装库名>

（3）pip3 list

（4）pip3 uninstall <拟卸载库名>

（5）pip3 show <库名>

（6）pip3 download <拟下载库名>

- Python常见的第三方库

库　名	用　途	库　名	用　途
numpy	矩阵运算、矢量处理、线性代数等	sklearn	机器学习和数据挖掘
matplotlib	2D&3D绘图库、数学运算、绘制图表	pyinstaller	Python源文件打包
PIL	通用的图像处理库	Django	支持快速开发的Web框架
requests	网页内容抓取	Scrapy	网页爬虫框架
jieha	中文分词	Flask	轻量级Web开发框架
bs4	HTML和XML解析	scipy	科学计算库
Wheel	Python文件打包	pandas	高效数据分析

- 使用pip安装第三方库，需要注意。

  - 在Python 3.x下，通常使用pip3命令安装，也可以使用pip命令安装。

  - 库名是第三方库常用的名字，pip安装用的文件名和库名不一定完全相同，通常采用小写字符。

  - 安装过程应在命令行下进行，而不是在IDLE中，部分库会依赖其他函数库，pip会自动安装，部分库下载后需要一个安装过程，pip也会自动执行。

7.6.3 pyinstaller库打包文件

- 用于源文件打包的第三方库，它能够在Windows、Linux等操作系统下将Python源文件打包。
- 打包后的Python文件可以在没有安装Python的环境中运行，也可以作为一个独立文件方便传递和管理。
- pyinstaller第三方库的官方网站网是：http://www.pyinstaller.org用户需要使用pip工具安装。

```
:\> pip3 install pyinstaller
```

- 使用PyInstaller库对Python源文件打包十分简单，使用方法如下：

  pyinstaller <Python源程序文件名>

- 执行完毕后，源文件所在目录将生成dist和build两个文件夹。最终的打包程序在dist内部与源文件同名的目录中。

- 可以通过-F参数对Python源文件生成一个独立的可执行文件，如下：

  PyInstaller -F <Python源程序文件名>

```
pyinstaller -F  xuexi.py
```

- 执行后在dist目录中出现了xuexi.exe文件，没有任何依赖库，执行它即可自动学习。

pyinstaller有一些常用参数

参数	功能
-h, --help	查看帮助
--clean	清理打包过程中的临时文件
-D, --onedir	默认值，生成dist目录
-F, --onefile	在dist文件夹中只生成独立的打包文件
-i <*图标文件名.ico*>	指定打包程序使用的图标（icon）文件