中国科学院大学
University of Chinese Academy of Sciences

计算机与控制学院
School of Computer and Control Engineering

移动互联网技术

第三章　互联网数据获取技术

Pyhon编程基础（一）

王文杰
wangwj@ucas.ac.cn

本节主要内容

- **概述**
- Python安装和使用
- Python的编码规范
- 变量和常量
- 数据类型
- 运算符与表达式
- 流程控制
- 函数
- Python常用内置函数
- 文件处理
- 目录的常见操作
- 面向对象编程
- 异常
- 数据库编程

概述

- 什么是**Python**?

- Python是一种解释型、面向对象的编程语言
- 是一个开源语言，拥有大量的库，可以高效地开发各种应用程序

概述

- **Python**主要特点
  - 简单易学
    - 设计哲学：优雅，明确，简单，可读性强
    - Python能让你专注于问题本身，而不是去搞明白语言本身

- **Python**格言

The Zen of Python

优美胜于丑陋
明了胜于晦涩
简洁胜于复杂
复杂胜于凌乱

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

by Tim Peters

概述

===C++===

```
#include<iostream>
using namespace std;

int main()
{
  cout<<"Hello World";
  return 0;
}
```

===Java===

```
public class Main{
  public static void main(String[] args)
  {
     System.out.println("Hello World");
  }
}
```

===Python===

```
print ("Hello World");
```

概述

- **Python**主要特点
  - 面向对象的高层语言
    - 完全支持继承、重载、派生、多继承
      - 但是消除了保护类型、抽象类、接口等面向对象的元素，使得面向对象的概念更容易理解。
    - 无需关注底层细节，而C/C++中需要指针操作
    - 与其他语言相比，Python以强大而有简单的方式实现面向对象编程
  - 脚本语言（Script Language）
    - 高级脚本语言，比脚本语言只能处理简单任务强大
    - Python与JavaScript、PHP、Perl等语言类似，它不需要另外声明变量、直接赋值即可创建一个新的变量

概述

- **Python主要特点**
  - 丰富的库：Python具有大量标准的、高质量的库
    - NumPy 基于Python的科学计算第三方库，提供了矩阵，线性代数，傅立叶变换等等的解决方案
    - Pandas（Python Data Analysis Library）是基于NumPy 的一种工具，该工具是为了解决数据分析任务而创建的（让以Numpy为中心的应用变得更加简单）
    - Matplotlib 用Python实现的类matlab的第三方库，用以绘制一些高质量的数学二维图形
    - SciPy 基于Python的matlab实现，旨在实现matlab的所有功能
    - Scapy是一个强大的python第三方网络数据包处理库，可以对网络数据流量，进行嗅探、分析、构建或者对网络进行攻击，如果有需要
    - Sklearn:
    - nltk

概述

- **Python**的诞生
  - 作者：Guido van Rossum，1991年公开版本发行
  - 名字来源：Monty Python's Flying Circus
  - 作者为什么发明Python：平衡C和Shell
  - 是当今最受欢迎的语言之一

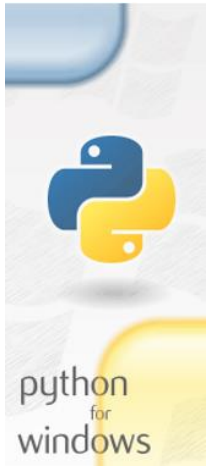
概述

- **Python**应的用途
  - 网页开发
  - 可视化（GUI）界面开发
  - 网络
  - 系统编程
  - 数据分析
  - 机器学习
  - 网络爬虫
  - 科学计算
  - 人工智能

本节主要内容

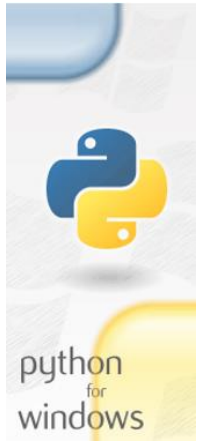
- 概述
- **Python安装和使用**
- Python的编码规范
- 变量和常量
- 数据类型
- 运算符与表达式
- 流程控制
- 函数
- Python常用内置函数
- 文件处理
- 目录的常见操作
- 面向对象编程
- 异常
- 数据库编程

Python安装



- 官网：https://www.python.org/
- **Python版本：**

Python编程环境



- 默认编程环境：IDLE---Python自带
- 其他常用开发环境：
  - Eclipse+PyDev---常用
  - pyCharm：社区版是免费的
  - Anaconda（内含Jupyter和Spyder）：即提供了Python，又提供了一些包
  - ……

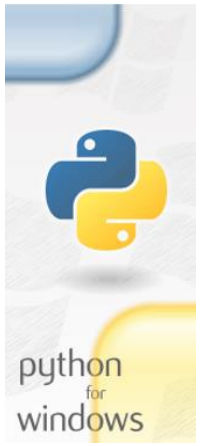
Python扩展库的安装

- Python 3.4以后的版本包含pip和setuptools库
  - pip用于安装管理Python扩展包
  - setuptools用于发布Python包
- pip的典型应用是从PyPI（Python Package Index：https://pypi.org/ ）上安装Python第三方包

  pip install lxml
- Pip也可以用清华的镜像，速度比较快

  pip install –i https://pypi.tuna.tsinghua.edu.cn/simple lxml

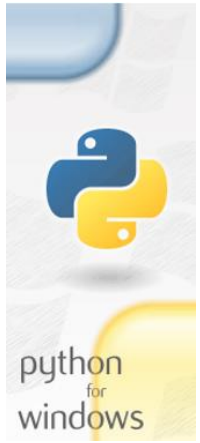
import 模块名 [as 别名]




```
>>> import math                          #导入标准库math
>>> math.sin(0.5)                        #求0.5（单位是弧度）的正弦
0.479425538604203
>>> import random                        #导入标准库random
>>> n = random.random()                  #获得[0,1) 内的随机小数
>>> n = random.randint(1,100)            #获得[1,100]区间上的随机整数
>>> n = random.randrange(1, 100)         #返回[1, 100)区间中的随机整数
>>> import os.path as path               #导入标准库os.path，并设置别名为path
>>> path.isfile(r'C:\windows\notepad.exe')
True
>>> import numpy as np                   #导入扩展库numpy，并设置别名为np
>>> a = np.array((1,2,3,4))              #通过模块的别名来访问其中的对象
>>> a
array([1, 2, 3, 4])
>>> print(a)
[1 2 3 4]
```

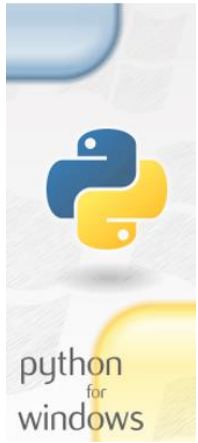
from 模块名 import 对象名[as 别名]

```
>>> from math import sin                #只导入模块中的指定对象，访问速度略快
>>> sin(3)
0.14112000805986722
>>> from math import sin as f           #给导入的对象起个别名
>>> f(3)
0.14112000805986722
>>> from os.path import isfile
>>> isfile(r'C:\windows\notepad.exe')
True
```



from 模块名 import *

```
>>> from math import *          #导入标准库math中所有对象
>>> sin(3)                      #求正弦值
0.1411200080598672
>>> gcd(36, 18)                 #最大公约数
18
>>> pi                          #常数π
3.141592653589793
>>> e                           #常数e
2.718281828459045
>>> log2(8)                     #计算以2为底的对数值
3.0
>>> log10(100)                  #计算以10为底的对数值
2.0
>>> radians(180)                #把角度转换为弧度
3.141592653589793
```



本节主要内容

- 概述
- Python安装和使用
- **Python的编码规范**
- 变量和常量
- 数据类型
- 运算符与表达式
- 流程控制
- 函数
- Python常用内置函数
- 文件处理
- 目录的常见操作
- 面向对象编程
- 异常
- 数据库编程

Python的编码规范

1、命名规则
2、代码缩进与冒号
3、使用空行分隔代码
4、注释
5、语句的分隔

1、命名规则

Python中标识符命名规则为：

- 第一个字符为字母或下划线
- 除第一个字符以外的其他字符可以是字母、下划线或数字
- 例如：

```
rulemodule.py                #模块名，即文件名
_rule='rule information'  #_rule变量名，
                             #通常前缀有一个下划线的变量名为全局变量
```

2、代码缩进与冒号

- 对于Python而言，代码缩进是一种语法。
- Python语言中没有采用花括号或begin…end分隔代码块，而是使用冒号和代码缩进区分代码之间的层次。

```
x=1
if x==1:
    print("x=",x)
else:
    print("x=",x)
    x=x+1
print("x=",x)
```

执行结果：

```
x= 1
x= 1
```

```
x=1
if x==1:
    print("x=",x)
else:
    print("x=",x)
x=x+1
print("x=",x)
```

执行结果：

```
x= 1
x= 2
```

- python程序是依靠代码块的缩进来体现代码之间的逻辑关系的，缩进结束就表示一个代码块结束了。
- 同一个级别的代码块的缩进量必须相同
- 一般，使用4个空格来表示每行的缩进

3、使用空行分隔代码

- 函数之间或类的方法之间用空行分隔，表示一段新的代码的开始。
- 类和函数入口之间也用一行空行分隔，突出函数入口的开始。

```
class A:
    def funX(self):
        print("funX()")

                                    #空行，便于阅读，表示区分方法之间的间隔
    def funY(self):
        print("funY()")

                                    #空行，下面是主程序入口
if __name__=="__main()__":
    a=A()
    a.funX()
    a.funY()
```

4、注释

- 在Python中，我们如果想让某些区域不起作用，我们可以对这些区域进行注释，常见的注释方法有：

  1、#注释法

  2、三引号注释法

- Python可以使用中文注释。
  - Python 3默认的编码是Unicode，可以直接使用中文注释；但在Python 2中若使用中文注释，必须在Python文件的最前面加上：

    ```
    # -*- coding: UTF-8 -*-
    ```

  - 同样，如果用print等语句打印中文相关的语句的时候，如果无法正常显示，也可以加入上述语句

5、语句的分隔

- 分号是C、Java等语言中标识语句结束的标志。Python也支持分号，同样可以用分号作为一行语句的结束标识。在C、Java中分号是必须的；而Python的分号可以省略，主要通过换行来识别语句的结束。
- 如果要在一行中书写多个语句，就必须使用分号了，否则Python无法识别语句之间的间隔。

```
#下面两条语句等价
print("Hello world!")
print("Hello world!");
```

执行结果：

```
Hello world!
Hello world!
```

```
#使用分号分隔符
x=1;y=1;z=1
print(x,y,z)
```

执行结果：

```
1 1 1
```

- **Python同样支持多行写一条语句，Python使用“\”作为换行符。多行写一条语句适用于长语句的情况。**

```
#字符串的换行
#写法一
sql="select id,name\
from dept\
where name='A'"
print(sql)

#写法二
sql="select id,name"\
      "from dept"\
      "where name='A'"
print(sql)
```

执行结果相同：

```
select id,namefrom deptwhere name='A'
select id,namefrom deptwhere name='A'
```

本节主要内容

- 概述
- Python安装和使用
- Python的编码规范
- **变量和常量**
- 数据类型
- 运算符与表达式
- 流程控制
- 函数
- Python常用内置函数
- 文件处理
- 目录的常见操作
- 面向对象编程
- 异常
- 数据库编程

变量和常量

1、变量命名规则和定义
2、变量如何引用值
3、多重赋值
4、局部变量和全局变量
5、常量

1、变量命名规则和定义

- 变量名的长度不受限制，其中的字符必须是字母、数字或下划线（_），不能使用空格、连字符、标点符号、引号或其他字符。
- 变量名的第一个字符不能是数字，必须是字母或下划线。
- Python区分大小写，因此TAX、Tax和tax是截然不同的变量名。
- 不能将Python关键字（或称为保留词）用作变量名。

1、变量命名规则和定义

- 在Python中，不需要事先声明变量名及其类型，直接赋值即可创建各种类型的对象变量。这一点适用于Python任意类型的对象。

例如语句

凭空出现一个整型变量x

```
>>> x = 3
```

创建了整型变量x，并赋值为3，再例如语句

新的字符串变量，再也不是原来的x了

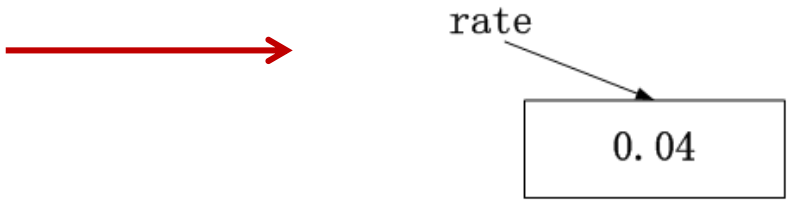
```
>>> x = 'Hello world.'
```

创建了字符串变量x，并赋值为’Hello world.’。

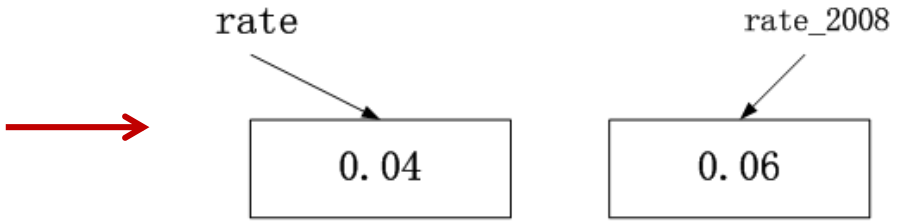
2、变量如何引用值

- **Python变量都需要先赋值，后引用：没有初始值**
- **对于x=expr这样的赋值语句，可以这样理解：让x指向表达式的值。**

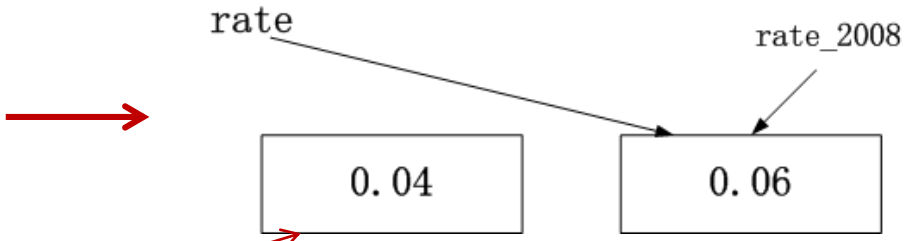
- rate=0.04



- rate_2008=0.06



- rate=rate_2008



Python自动将其删除，称为垃圾收集

2、变量如何引用值

- Python采用的是基于值的内存管理方式，如果为不同变量赋值为相同值（仅适用于-5至256的整数和短字符串），这个值在内存中只有一份，多个变量指向同一块内存地址。

```
>>> x = 3
>>> id(x)
10417624
>>> y = 3
>>> id(y)
10417624
>>> x = [1, 1, 1, 1]
>>> id(x[0]) == id(x[1])
True
```

id()函数用来查看对象的内存地址

3、多重赋值

- **Python**中，有一种便利的方法，能够同时给多个变量赋值。

```
>>> x,y,z=1,'two',3.0
>>> x
1
>>> y
'two'
>>> z
3.0
>>> x,y,z
(1, 'two', 3.0)
```

- 将两个变量的值互换的标准方式（其他高级语言均如此）为：

```
>>> a,b=5,9
>>> temp=a
>>> a=b
>>> b=temp
>>> a,b
(9, 5)
```

```
>>> a,b=5,9
>>> a,b
(5, 9)
>>> a,b=b,a
>>> a,b
(9, 5)
```

使用多重赋值实现两个变量的值互换。

4、局部变量和全局变量

（1）局部变量

- 局部变量是只能在函数或代码块内使用的变量。
- 函数或代码段一旦结束，局部变量的生命周期也就结束。
- 局部变量的作用范围只在其被创建的函数内有效。

```
#局部变量
def fun():
    local=1
    print(local)

fun()

print(local)
```

此时已超出了local变量的作用范围

运行结果：

```
1
Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    print(local)
NameError: name 'local' is not defined
```

4、局部变量和全局变量

（2）全局变量

- 全局变量通常在文件的开始处定义。

```
#在文件的开头定义全局变量
_a=1                                    #定义全局变量_a
_b=2                                    #定义全局变量_b
def add():                              #定义加法函数
    global _a                           #用global引用全局变量
    _a=3                                #给全局变量_a重新赋值
    return "_a+_b=",_a+_b               #返回加法结果
def sub():                              #定义减法函数
    global _b
    _b=4
    return "_a-_b=",_a-_b

print(add())
print(sub())
```

运行结果：

```
('_a+_b=', 5)
('_a-_b=', -1)
```

全局变量使用注意事项：

- 可以将全局变量放到一个专门的文件中，便于统一管理。

- 全局文件名：**gl.py**

```
#全局变量
_a=1
_b=2
```

```
#调用全局变量
import gl                #导入之前创建的模块gl
def fun():               #定义函数fun()，调用全局变量_a和_b
    print(gl._a)         #此处不需要使用global关键字，因为使用了前导符gl定位全局变量
    print(gl._b)

fun()
```

运行结果：

```
1
2
```

5、常量

- 常量是一旦初始化后就不能改变的量。
- 例如：数字5、字符串”abc”都是常量。
- True、False、None

本节主要内容

- 概述
- Python安装和使用
- Python的编码规范
- 变量和常量
- **数据类型**
- 运算符与表达式
- 流程控制
- 函数
- Python常用内置函数
- 文件处理
- 目录的常见操作
- 面向对象编程
- 异常
- 数据库编程

数据类型

1. 数字
2. 序列的概念
3. 字符串
4. 元组tuple
5. 列表list
6. 字典dict
7. 集合set

容器：数据结构

对象类型	类型名称	示例	简要说明
数字	int float complex	1234 3.14, 1.3e5 3+4j	数字大小没有限制，内置支持复数及其运算
字符串	str	'swfu', "I'm student", '''Python ''', r'abc', R'bcd'	使用单引号、双引号、三引号作为定界符，以字母r或R引导的表示原始字符串
字节串	bytes	b'hello world'	以字母b引导，可以使用单引号、双引号、三引号作为定界符
列表	list	[1, 2, 3] ['a', 'b', ['c', 2]]	所有元素放在一对方括号中，元素之间使用逗号分隔，其中的元素可以是任意类型
字典	dict	{1:'food' ,2:'taste', 3:'import'}	所有元素放在一对大括号中，元素之间使用逗号分隔，元素形式为“键:值”
元组	tuple	(2, -5, 6) (3,)	不可变，所有元素放在一对圆括号中，元素之间使用逗号分隔，如果元组中只有一个元素的话，后面的逗号不能省略
集合	set frozenset	{'a', 'b', 'c'}	所有元素放在一对大括号中，元素之间使用逗号分隔，元素不允许重复;另外，set是可变的，而frozenset是不可变的
布尔型	bool	True, False	逻辑值，关系运算符、成员测试运算符、同一性测试运算符组成的表达式的值一般为True或False
空类型	NoneType	None	空值

数据类型

- Python 3中，一切皆为对象
  - 每个对象由标识（identity）、类型（type）和值（value）标识
- 标识（identity）用于唯一标识一个对象，通常对应于对象在计算机内存中的位置。使用内置函数id(obj1)可返回对象obj1的标识
- 类型（type）用于表示对象所属的数据类型。使用内置函数type(obj1)可以返回对象obj1所属的数据类型
- 值（value）用于表示对象的数据类型的值。

  使用内置函数print(obj1)可返回对象obj1的值

```
>>> i=1
>>> print(type(i))
<class 'int'>
>>> f=1.2
>>> print(type(f))
<class 'float'>
```

1、数字

- **Python 3的数字类型分为：**
  - 整型：**unlimited length**
  - 浮点型：实现用**double in C。**
    可查看 **sys.float_info**
  - 布尔型
  - 复数类型： 复数是个对象，创建时需要用构造函数**complex**
    real(实部) & imaginary(虚部）
    用**z.real** 和 **z.imag**来取两部分

复数运算：

- Python内置支持复数类型及其运算，并且形式与数学上的复数完全一致。

```
>>> x = 3 + 4j                          #使用j或J表示复数虚部
>>> y = 5 + 6j
>>> x + y                               #支持复数之间的加、减、乘、除以及幂乘等运
算
(8+10j)
>>> x * y
(-9+38j)
>>> abs(x)                              #内置函数abs()可用来计算复数的模
5.0
>>> x.imag                              #虚部
4.0
>>> x.real                              #实部
3.0
>>> x.conjugate()                       #共轭复数
(3-4j)
```

2、序列的概念

- 在Python中，序列是一组按顺序排列的值。
- Python有3种内置序列类型：字符串、元组和列表。
  - 字符串和列表是最常见的序列，元组较少使用。
- 所有序列的特征：
  - 第一个正索引为零，指向左端；
  - 第一个负索引为-1，指向右端；
  - 可使用分片表示法来复制子序列；
  - 可使用+和*进行拼接（即合并）。进行拼接的序列类型必须相同；
  - 可使用函数len计算其长度；
  - 表达式x in s检查序列s是否包含元素x，如果x位于s中，则返回True，否则返回False

3、字符串

- 字符串由一系列字符组成。
- 在Python中，可以使用三种方式表示字符串。
  - 单引号，如： 'http'、'open windows'、 'cat'
  - 双引号，如： "http"、 "open windows"、 "cat"
  - 三引号，如： """ http"""或多行字符串：

```
"""
Me and my monkey
have something to hide
"""“
```

说明：

- 三种引号是等价的。
- 单引号和双引号的一个主要用途为：可以在字符串中包含字符"和'。如：

```
>>> print("It's great")
It's great
>>> print('She said "Yes!"')
She said "Yes!"
```

- 三引号适用于创建多行字符串。三引号括起的字符串中还可以包含字符"和'

```
>>> print("Hello world!")
Hello world!
>>> print('Hello World!')
Hello World!
>>> print('''This is the first line
              This ia the second line
              Lastline''')
This is the first line
              This ia the second line
              Lastline
>>> print("""This is the first line
             This is the second line
            Last line""")
This is the first line
             This is the second line
             Last line
```

字符串的几个函数

（1）求字符串的长度： len(x)

- 由于函数len返回一个整数，所以在任意可以使用整数的地方，都可以使用len。
- 例如：

```
>>> len('pear')
4
>>> len('up, up, and away')
16
>>> len("moose")
5
>>> len("")
0
>>> 5+len('cat')*len('dog')
14
```

字符串的几个函数

（2）字符串拼接

- 用加号（+或*）拼接字符串

```
>>> 'hot'+'dog'
'hotdog'
>>> 'Once'+" "+'Upon'+' '+"a Time"
'Once Upon a Time'
```

将同一个字符串拼接多次，可使用如下快捷方式：

```
>>> 10*'ha'
'hahahahahahahahahahaha'
>>> 3*'hee'+2*'!'
'heeheehee!!'
>>> len(2*'pizza pie!')
20
>>> len("house"+'boat')*'ab'
'abababababababababab'
```

字符串的几个函数

- 用print()函数拼接字符串
- 当在一条语句中输出多个字符串时，print()函数会自动地插入空格，只需用逗号将不同的字符串隔开即可。例如：

```
>>> print("John","Everyman")
John Everyman
```

- 换行

```
>>> print("What's your name? \nTom")
What's your name?
Tom
```

字符串的几个函数

- **Format字符串**

```
>>> age=3
>>> name="Tom"
>>> print("{0} was {1} years old".format(name, age))
Tom was 3 years old
```

- 也可以用下面方法实现

```
>>> print(name + " was " + str(age) + " years old")
Tom was 3 years old
```

字符串与字节串

对str类型的字符串调用其encode()方法进行编码得到bytes字节串，对bytes字节串调用其decode()方法并指定正确的编码格式则得到str字符串。

```
>>> type('Hello world')                                    #默认字符串类型为str
<class 'str'>
>>> type(b'Hello world')                                   #在定界符前加上字母b表示字节串
<class 'bytes'>
>>> '中国科学院大学'.encode('utf8')                        #使用utf-8对中文进行编码
b'\xe4\xb8\xad\xe5\x9b\xbd\xe7\xa7\x91\xe5\xa
d\xa6\xe9\x99\xa2\xe5\xa4\xa7\xe5\xad\xa6'
>>> _.decode('utf8')                                      #一个下划线表示最后一次正确输出结果
'中国科学院大学'
```

4、元组

- 元组是一种不可变序列，即创建之后不能再做任何修改。
- 元组由不同的元素组成，每个元素可以存储不同类型的数据，
  - 如字符串、数字甚至元组。
  - 元组通常代表一行数据，而元组中的元素代表不同的数据项。

```
>>> item2=('cat',-6,(1,2))
>>> print(item2)
('cat', -6, (1, 2))
>>> print(type(item2))
<class 'tuple'>
```

（1）元组的创建

- tuple=(元素1,元素2,…元素n)　#定义n个元素组成的元组

  tuple=()　　#定义空元组

  tuple=(元素1,)　　#定义单元素元组，需要用逗号结尾消除歧义

```
>>> item3=()
>>> print(item3)
()
>>> print(type(item3))
<class 'tuple'>
```

```
>>> item1=(4,)
>>> print(item1)
(4,)
>>> print(type(item1))
<class 'tuple'>
```

```
>>> item4=(4)
>>> print(item4)
4
>>> print(type(item4))
<class 'int'>
```

（2）元组的访问

- 元组中元素的值通过索引访问，索引也称为“下标”。
- 格式：

tuple[n] **#访问第n个元素**

tuple[m:n] **#访问第1个索引到第2个索引之间的索引元素，**
但不包括第2个索引指向的元素

其中，n、m可以为0、正、负整数。

```
fruit=('aaple','banana','grape','orange')
print(fruit[0],fruit[1])
print(fruit[-1])
print(fruit[-2])
fruit1=fruit[1:3]  #切片。从第2个元素到第3个元素，不包括第4个
fruit2=fruit[0:-2] #切片。从第1个元素到倒数第2个元素，不包括倒数第2个
fruit3=fruit[2:-2] #切片。从第3个元素到倒数第2个元素，不包括倒数第2个
print(fruit1)
print(fruit2)
print(fruit3)
```

```
aaple banana
orange
grape
('banana', 'grape')
('aaple', 'banana')
()
```

（3）操作元组

```
t1=(1,'two',3)
t2=(t1,'four')

print(t1+t2)          #连接
print((t1+t2)[3])     #索引
print((t1+t2)[2:5])#分片

t3=('five',)          #单元素元组
print(t1+t2+t3)
```

```
---
(1, 'two', 3, (1, 'two', 3), 'four')
(1, 'two', 3)
(3, (1, 'two', 3), 'four')
(1, 'two', 3, (1, 'two', 3), 'four', 'five')
```

常用元组函数

函数名	返回值
x in tupple	若x是元组tupple中的一个元素，则返回True，否则返回False
len(tupple)	元组tupple包含的元素数
tupple.count(x)	元素x在元组tupple中出现的次数
tupple.index(x)	元组tupple中第一个元素x的索引，若x未包含在元组tupple中，将引发ValueError异常

元组函数应用

```
pets=('dog','cat','bird','dog')
print(pets)
for i in range(len(pets)): #遍历元组
    print(pets[i])
print('bird' in pets)
print('cow' in pets)
print(pets.count('dog'))
print(pets.index('dog'))
print(pets.index('mouse'))
```

```
('dog', 'cat', 'bird', 'dog')
dog
cat
bird
dog
True
False
2
0
ValueError: tuple.index(x): x not in tuple
```

5、列表List—就是数组

- 列表是Python中非常重要的数据类型，通常作为函数的返回类型。
- 列表和元组相似，也是由一组元素组成。列表可包含任何类型的值：数字、字符串甚至序列。
- 列表与元组的重要差别是列表是可变的，即可以在不复制的情况下添加、删除或修改列表元素。

列表的创建

- 格式：

  list=[元素1,元素2,...元素n]　#定义n个元素组成的列表

  list=[]　　#定义空列表

  list=[x]　　#定义只包含一个元素的列表，

  　　与元组不同，最后的逗号不是必须的

- 说明：列表用方括号括起，其中元素用逗号分隔

```
numbers1=[7,-7,2,3,2]
print(numbers1)
print(type(numbers1))
numbers2=[7]
print(numbers2)
print(type(numbers2))
numbers3=[]
print(numbers3)
print(type(numbers3))
```

```
[7, -7, 2, 3, 2]
<class 'list'>
[7]
<class 'list'>
[]
<class 'list'>
```

列表的使用

- 列表的使用与元组十分相似，同样支持负数索引、分片以及多元列表等特性，但列表的元素可修改。
- 与字符串和元组一样，可使用len获取列表长度，还可使用+和*拼接列表。
- 常用列表函数

函数	返回值
s.append(x)	在列表s末尾处添加元素x
s.count(x)	返回元素x在列表中出现的次数
s.extend(lst)	将lst的所有元素都添加到列表s末尾
s.index(x)	返回第一个x元素的索引
s.insert(i,x)	将元素x插入到索引i指定的元素前面，结果是s[i]=x
s.pop(x)	删除并返回s中索引为i的元素
s.remove(x)	删除s中的第一个x元素
s.reverse(x)	反转s中元素的排列顺序
s.sort()	将s的元素按升序排列

函数作为列表的元素

- 用函数作为参数与列表一起使用非常有用，也称为高阶编程。
- 例：假定L是一个列表，f是一个函数，用函数替换每个元素改变列表，即用f(e)改变L中的元素e。

执行结果：

```
[1, 2, 3.4]
[1, 2, 3]
[1, 2, 6]
[1, 2, 13]
```

```
def applyToEach(L,f):
    for i in range(len(L)):
        L[i]=f(L[i])

L=[1,-2,3.4]

def fact(n):
    if n==1:
        return 1
    else:return n*fact(n-1)

def fib(n):
    if n==0 or n==1:
        return 1
    else:return fib(n-1)+fib(n-2)

applyToEach(L, abs)
print(L)
applyToEach(L, int)
print(L)
applyToEach(L, fact)
print(L)
applyToEach(L, fib)
print(L)
```

6、字典

- 字典是Python重要的数据类型，字典是由“键—值”对组成的集合，字典中的“值”通过“键”来引用。
- 字典也称为关联数组、映射或散列表。
- 列表一样，字典也是可以改变的：
  – 可以添加、删除或修改“键—值”对
- 使用“键”来访问字典值效率极高。

创建字典

- 格式：

  dictionary={key1:value1, key2:value2, …, keyn:valuen)}

  #创建n个“键—值”对组成的字典

  dictionar={}　　#创建空字典

- 注意：
  - 字典中的键必须独一无二，即在同一个字典中，任何两个键—值对都不能相同；
  - 键必须是不可变的。对值没有这两个限制。

字典的访问

- 字典的访问与元组、列表有所不同，元组和列表是通过数字索引获取对应的值，而字典是通过key值获取相应的value值。
- 格式： value=dict[key]
- 说明：
  - 字典的添加、删除和修改只需执行一条赋值语句即可，例如：dict['x']='value'
  - 字典没有remove操作。删除字典元素可调用内置函数del()完成。

字典的访问

```
dict={'a':'apple', 'b':'banana', 'g':'grape', 'o':'orange'} #创建字典
print(dict)
dict['w']='watermelon'  #添加字典元素
print(dict)
del(dict['a']) #删除字典中键为'a'的元素
print(dict)
dict['g']='grapefruit' #修改字典中键为'g'的值
print(dict)
```

```
{'a': 'apple', 'b': 'banana', 'g': 'grape', 'o': 'orage'}
{'a': 'apple', 'b': 'banana', 'g': 'grape', 'o': 'orage', 'w': 'watermelon'}
{'b': 'banana', 'g': 'grape', 'o': 'orage', 'w': 'watermelon'}
{'b': 'banana', 'g': 'grapefruit', 'o': 'orage', 'w': 'watermelon'}
```

7、集合

- 在Python中，集合是一系列不重复的元素。
- 集合类似于字典，但只包含键，而没有相关联的值。
- 集合分两类：可变集合（set）和不可变集合（frozenset）。
  - 对于可变集合，可添加和删除元素，
  - 而不可变集合一旦创建就不能更改。
- 与字典一样，集合的元素排列顺序也是不确定的。

集合基本功能

- 包括关系测试和消除重复元素。
- 集合对象还支持 **union**（联合），**intersection**（交），**difference**（差）和 **sysmmetric difference**（对称差集，即异或）等数学运算。
- 注意：想要创建空集合，必须使用 **set()** 而不是 { } 。

集合基本功能

```
basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
print(basket)
print(type(basket))
fruit = set(basket)    # create a set without duplicates
print(fruit)
print(type(fruit))

print('orange' in fruit )     # fast membership testing
print('crabgrass' in fruit)
```

执行结果：

```
['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
<class 'list'>
{'apple', 'orange', 'pear', 'banana'}
<class 'set'>
True
False
```

集合操作

```
a = set('abracadabra')
b = set('alacazam')
print(a)    # unique letters in a
print(b)    # unique letters in a

#差运算
print(a - b)     # letters in a but not in b

#或运算
print(a | b)     # letters in either a or b

#并运算
print(a & b)     # letters in both a and b

#异或运算
print(a ^ b)     # letters in a or b but not both
```

执行结果：

```
{'a', 'b', 'r', 'c', 'd'}
{'a', 'l', 'z', 'c', 'm'}
{'b', 'r', 'd'}
{'b', 'z', 'c', 'd', 'm', 'r', 'a', 'l'}
{'a', 'c'}
{'b', 'z', 'd', 'm', 'r', 'l'}
```

数据类型小结

	列表	元组	字典	集合
类型名称	list	tuple	dict	set
定界符	方括号[]	圆括号()	大括号{}	大括号{}
是否可变	是	否	是	是
是否有序	是	是	否	否
是否支持下标	是（使用序号作为下标）	是（使用序号作为下标）	是（使用“键”作为下标）	否
元素分隔符	逗号	逗号	逗号	逗号
对元素形式的要求	无	无	键:值	必须可哈希
对元素值的要求	无	无	“键”必须可哈希	必须可哈希
元素是否可重复	是	是	“键”不允许重复，“值”可以重复	否
元素查找速度	非常慢	很慢	非常快	非常快
新增和删除元素速度	尾部操作快 其他位置慢	不允许	快	快

数据类型小结

```
>>> x_list = [1, 2, 3]                          #创建列表对象
>>> x_tuple = (1, 2, 3)                         #创建元组对象
>>> x_dict = {'a':97, 'b':98, 'c':99}           #创建字典对象
>>> x_set = {1, 2, 3}                           #创建集合对象
>>> print(x_list[1])                            #使用下标访问指定位置的元素
2
>>> print(x_tuple[1])                           #元组也支持使用序号作为下标
2
>>> print(x_dict['a'])                          #字典对象的下标是“键”
97
>>> 3 in x_set                                  #成员测试
True
```

本节主要内容

- 概述
- Python安装和使用
- Python的编码规范
- 变量和常量
- 数据类型
- **运算符与表达式**
- 流程控制
- 函数
- Python常用内置函数
- 文件处理
- 目录的常见操作
- 面向对象编程
- 异常
- 数据库编程

运算符与表达式

运算符	功能说明
+	算术加法，列表、元组、字符串合并与连接，正号
-	算术减法，集合差集，相反数
*	算术乘法，序列重复
/	真除法
//	求整商，但如果操作数中有实数的话，结果为实数形式的整数
%	求余数，字符串格式化
**	幂运算
<、<=、>、>=、==、!=	（值）大小比较，集合的包含关系比较
or	逻辑或
and	逻辑与
not	逻辑非
in	成员测试
is	对象同一性测试，即测试是否为同一个对象或内存地址是否相同
|、^、&、<<、>>、~	位或、位异或、位与、左移位、右移位、位求反
&、|、^	集合交集、并集、对称差集
@	矩阵相乘运算符

1. 算术运算符

- （1）+运算符除了用于算术加法以外，还可以用于列表、元组、字符串的连接，但不支持不同类型的对象之间相加或连接。

```
>>> 'A' + 1                          #不支持字符与数字相加，抛出异常
TypeError: Can't convert 'int' object to str implicitly
>>> True + 3                         #Python内部把True当作1处理
4
>>> False + 3                        #把False当作0处理
3
```

1. 算术运算符

（2）*运算符除了表示算术乘法，还可用于列表、元组、字符串这几个序列类型与整数的乘法，表示序列元素的重复，生成新的序列对象。字典和集合不支持与整数的相乘，因为其中的元素是不允许重复的。

```
>>> True * 3
3
>>> False * 3
0
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> (1, 2, 3) * 3
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> 'abc' * 3
'abcabcabc'
```

1. 算术运算符

（3）运算符/和//在Python中分别表示算术除法和算术求整商（floor division）。

```
>>> 3 / 2                #数学意义上的除法
1.5
>>> 15 // 4              #如果两个操作数都是整数，结果为整数
3
>>> 15.0 // 4            #如果操作数中有实数，结果为实数形式的整数值
3.0
>>> -15//4               #向下取整
-4
```

（4）%运算符可以用于整数或实数的求余数运算，还可以用于字符串格式化，但是这种用法并不推荐

1. 算术运算符

（5）**运算符表示幂乘：

```
>>> 3 ** 2                    #3的2次方，等价于pow(3, 2)
9
>>> pow(3, 2, 8)              #等价于(3**2) % 8
1
>>> 9 ** 0.5                  #9的0.5次方，平方根
3.0
>>> (-9) ** 0.5               #可以计算负数的平方根
(5.809009821810581e-17+0.9486832980505138j)
```

2. 关系运算符

- Python关系运算符最大的特点是可以连用

```
>>> 1 < 3 < 5          #等价于1 < 3 and 3 < 5
True
>>> 1 > 6 < 8
False
>>> {1, 2, 3} < {1, 2, 3, 4}          #测试是否子集
True
>>> {1, 2, 3} == {3, 2, 1}          #测试两个集合是否相等
True
```

成员测试运算符in与同一性测试运算符is

- 成员测试运算符in用于成员测试，即测试一个对象是否为另一个对象的元素

```
>>> 3 in [1, 2, 3]                    #测试3是否存在于列表[1, 2, 3]中
True
```

成员测试运算符in与同一性测试运算符is

- 同一性测试运算符is用来测试两个对象是否是同一个，如果是则返回True，否则返回False。
- 如果两个对象是同一个，二者具有相同的内存地址。

```
>>> 3 is 3
True
>>> x = [300, 300, 300]
>>> x[0] is x[1]                    #基于值的内存管理，同一个值在内存中只有一份
True
>>> x = [1, 2, 3]
>>> y = [1, 2, 3]
>>> x is y                          #上面形式创建的x和y不是同一个列表对象
False
>>> x[0] is y[0]
True
```

3. 位运算符与集合运算符

- 位运算符只能用于整数，其内部执行过程为：
  - 首先将整数转换为二进制数，然后右对齐，必要的时候左侧补0，按位进行运算，最后再把计算结果转换为十进制数字返回
  - 位与运算规则为1&1=1、1&0=0&1=0&0=0，
  - 位或运算规则为1|1=1|0=0|1=1、0|0=0，
  - 位异或运算规则为1^1=0^0=0、1^0=0^1=1。
  - 左移位时右侧补0，每左移一位相当于乘以2；
  - 右移位时左侧补0，每右移一位相当于整除以2。

```
>>> 3 << 2      #把3左移2位
12
>>> 3 & 7       #位与运算
3
>>> 3 | 8       #位或运算
11
>>> 3 ^ 5       #位异或运算
6
```

3. 位运算符与集合运算符

- 集合的交集、并集、对称差集等运算借助于位运算符来实现，而差集则使用减号运算符实现（注意，并集运算符不是加号）。

```
>>> {1, 2, 3} | {3, 4, 5}          #并集，自动去除重复元素
{1, 2, 3, 4, 5}
>>> {1, 2, 3} & {3, 4, 5}          #交集
{3}
>>> {1, 2, 3} ^ {3, 4, 5}          #对称差集
{1, 2, 4, 5}
>>> {1, 2, 3} - {3, 4, 5}          #差集
{1, 2}
```

4. 逻辑运算符

- 逻辑运算符and、or、not常用来连接条件表达式构成更加复杂的条件表达式，并且and和or具有惰性求值或逻辑短路的特点，当连接多个表达式时只计算必须要计算的值
- 另外要注意的是，运算符and和or并不一定会返回True或False，而是得到最后一个被计算的表达式的值，但是运算符not一定会返回True或False。

4. 逻辑运算符

```
>>> 3>5 and a>3                    #注意，此时并没有定义变量a
False
>>> 3>5 or a>3                     #3>5的值为False，所以需要计算后面表达式
NameError: name 'a' is not defined
>>> 3<5 or a>3                     #3<5的值为True，不需要计算后面表达式
True
>>> 3 and 5>2
True
>>> 3 not in [1, 2, 3]            #逻辑非运算not
False
>>> 3 is not 5                     #not的计算结果只能是True或False之一
True
>>> not 3
False
>>> not 0
True
```

4. 逻辑运算符

- 可以作为条件表达式

```
>>> 3 and 5
5
>>> 3 or 5
3
>>> 0 and 5
0
>>> 0 or 5
5
>>> not 3
False
>>> not 0
True
```

5. 矩阵乘法运算符@

- 从Python 3.5开始增加了一个新的矩阵相乘运算符@，不过由于Python没有内置的矩阵类型，所以该运算符常与扩展库numpy一起使用。另外，@符号还可以用来表示修饰器的用法。

```
>>> import numpy                    #numpy是用于科学计算的Python扩展库
>>> x = numpy.ones(3)               #ones()函数用于生成全1矩阵，参数表示矩阵大小
>>> m = numpy.eye(3)*3              #eye()函数用于生成单位矩阵
>>> m[0,2] = 5                      #设置矩阵指定位置上元素的值
>>> m[2, 0] =3
>>> x @ m                           #矩阵相乘
array([ 6.,  3.,  8.])
```

本节主要内容

- 概述
- Python安装和使用
- Python的编码规范
- 变量和常量
- 数据类型
- 运算符与表达式
- **流程控制**
- 函数
- Python常用内置函数
- 文件处理
- 目录的常见操作
- 面向对象编程
- 异常
- 数据库编程

流程控制

1. 条件语句
2. 循环语句
3. 跳转语句

- Python没有switch
  - 只能用if等嵌套
- 只有while-do， 没有do-while
  - 要先判断条件，在去执行

注意加：！！

条件表达式

- 在选择和循环结构中，条件表达式的值只要不是False、0（或0.0、0j等）、空值None、空列表、空元组、空集合、空字典、空字符串、空range对象或其他空迭代对象，Python解释器均认为与True等价
- 在Python语法中，条件表达式中不允许使用赋值运算符“=”

```
>>> if a=3:
SyntaxError: invalid syntax
>>> if (a=3) and (b=4):
SyntaxError: invalid syntax
```

条件语句

1、if/else语句

（1）单分支

if(表达式):

语句序列

```
x = input('Input two number:')
a, b = map(int, x.split())
if a > b:
    a, b = b, a
print(a, b)
```

（2）双分支

if(表达式):

语句序列1

else:

语句序列2

```
chTest = ['1', '2', '3', '4', '5']
if chTest:
        print(chTest)
else:
        print('Empty')
```

条件语句

2、if...elif...else语句

- 假设航空公司提供了儿童优惠票价：不超过2岁的儿童免票；2-13岁的儿童打折；13岁及以上儿童与成人同价。

```
age=int(input('How old are you? '))
if age<=2:
    print('free')
elif 2<age<13:
    print('child fare')
else:
    print('adult fare')
```

```
How old are you? 2
free
>>> ==================
>>>
How old are you? 5
child fare
>>> ==================
>>>
How old are you? 15
adult fare
```

条件语句

3、if语句嵌套

```
if 表达式1:
    语句块1
    if 表达式2:
        语句块2
    else:
        语句块3
else:
    if 表达式4:
        语句块4
```

注意：缩进必须要正确并且一致

```
a=int(input('Input a:'))
b=int(input('Input b:'))
c=int(input('Input c:'))
if a<b:
    if b<c:
        max=c
    else:
        max=b
else:
    if a<c:
        max=c
    else:
        max=a
print('max=',max)
```

```
Input a:3
Input b:9
Input c:1
max= 9
```

条件语句

Python还提供了一个三元运算符

```
value1 if condition else value2
```

- **当条件表达式condition的值与True等价时，表达式的值为value1，否则表达式的值为value2。**

```
a=int(input('Input a:'))
b=int(input('Input b:'))
max=a if a>b else b
print('max=',max)
```

循环语句

- **两种循环结构的完整语法形式分别为：**

```
while 条件表达式:
    循环体
[else:
    else子句代码块]
```

```
for 取值 in 序列或迭代对象:
    循环体
[else:
    else子句代码块]
```

循环语句

- **for循环**
  - 例子：

```
a_list = ['a', 'b', 'mpilgrim', 'z', 'example']
for i, v in enumerate(a_list):
    print('列表的第', i+1, '个元素是：', v)
```

```
列表的第 1 个元素是： a
列表的第 2 个元素是： b
列表的第 3 个元素是： mpilgrim
列表的第 4 个元素是： z
列表的第 5 个元素是： example
```

循环语句：

- 说明：
  - For循环只作用于容器！！！，没有下述写法：

```
for(i=0; i<100; ++i):
        #todo
```

  - for循环通常与range()函数一起使用，range()函数返回一个列表，for循环遍历列表中的元素。
  - range()函数格式：range(start,stop[,step])，参数start表示列表开始值，默认为0；参数stop表示列表结束值，不能缺省，循环到stop-1停止；参数step表示步长，默认值为1。

循环语句

- **while循环**
  - **例子：求1+2+3+...+100**

```
i=1
s=0
while i<=100:
    s+=i
    i=i+1
print("sum=",s)
```

跳转语句

- Python的跳转语句有：break语句和continue语句。
  - 一旦break语句被执行，将使得break语句所属层次的循环提前结束；
  - continue语句的作用是提前结束本次循环，忽略continue之后的所有语句，提前进入下一次循环。

跳转语句

- 计算小于100的最大素数。

```
for n in range(100, 1, -1):
    if n%2 == 0:
        continue
    for i in range(3, int(n**0.5)+1, 2):
        if n%i == 0:
            #结束内循环
            break
    else:
        print(n)
        #结束外循环
        break
```

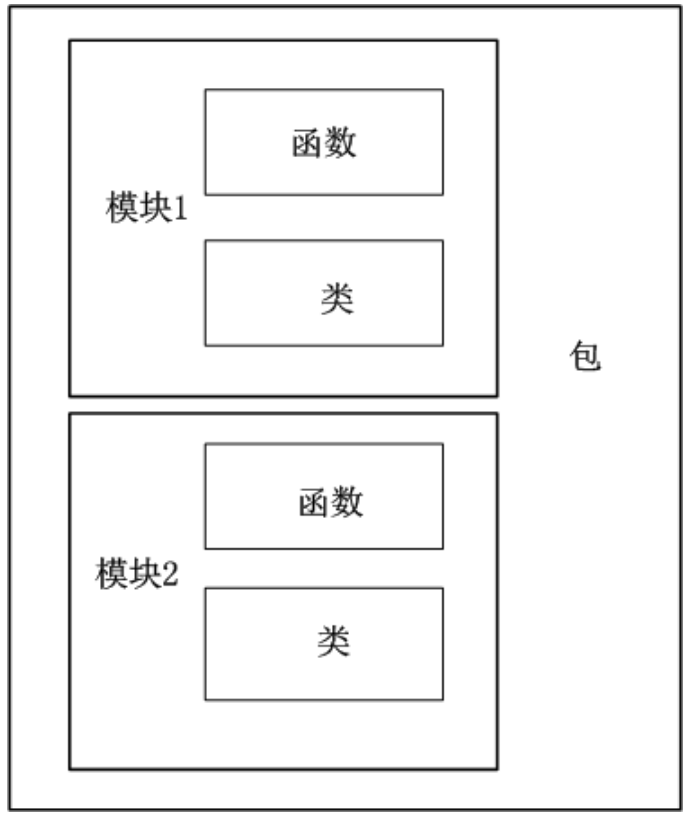
本节主要内容

- 概述
- Python安装和使用
- Python的编码规范
- 变量和常量
- 数据类型
- 运算符与表达式
- 流程控制
- **函数**
- Python常用内置函数
- 文件处理
- 目录的常见操作
- 面向对象编程
- 异常
- 数据库编程

模块与函数

- Python的程序由包、模块和函数组成。
- 函数是一段可重用的有名称的代码。通过输入的参数值，返回需要的结果，并可存储在文件中供以后使用。
- 模块是处理某一类问题的集合，模块由函数和类组成。模块和常规Python程序之间的唯一区别是用途不同：模块用于编写其他程序。因此，模块通常没有main函数。
- 包是一个完成特定任务的工具箱，Python提供了许多有用的工具包，如字符串处理、图形用户接口、Web应用、图像处理等。使用自带的工具包，可以提高程序开发效率、减少编程复杂度，达到代码重用的效果。

- Python的程序结构：



说明

- Python自带的工具包和模块安装在其安装目录的Lib子目录中。
  - 例如：Lib目录中的xml文件夹。xml文件夹就是一个包，该包用于完成XML的应用开发，xml包中包含四个子包：dom、sax、etree和parsers。文件__init__.py是xml包的注册文件，若无此文件，Python将不能识别xml包。
  - 注意：包必须至少含有一个__init__.py文件。 __init__.py文件的内容可以为空，它用于标识当前文件夹是一个包。

1、函数的定义及调用

- 函数定义语法：

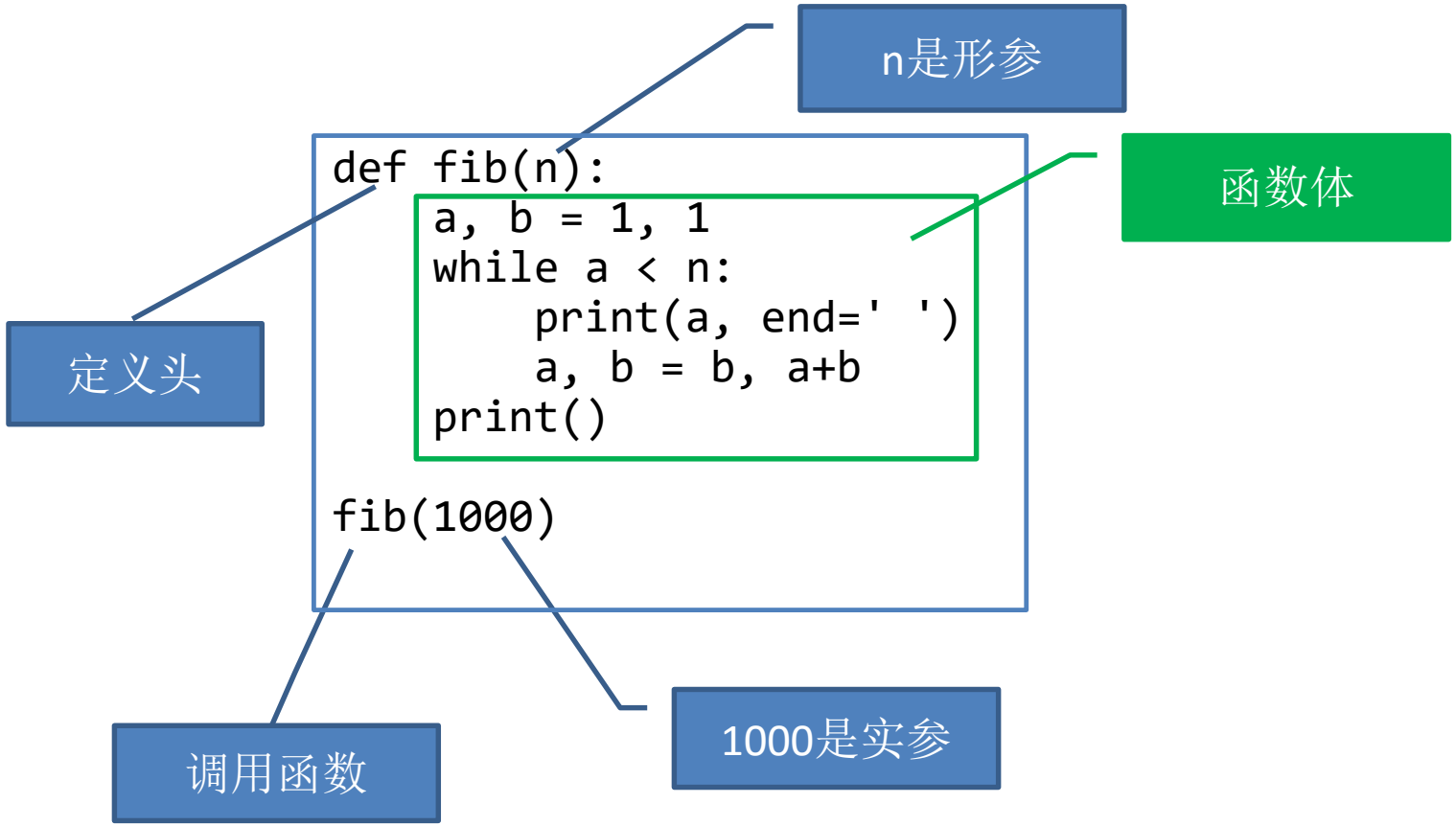
```
def 函数名([参数列表]):
    函数体

    [return 表达式]  #可选项，即有的函数可以没有返回值。
```

- 注意事项
  - 函数形参不需要声明类型，也不需要指定函数返回值类型
  - 括号后面的冒号必不可少
  - 函数体相对于def关键字必须保持一定的空格缩进
  - Python允许嵌套定义函数

- 在Python中，定义函数时也不需要声明函数的返回值类型，而是使用return语句结束函数执行的同时返回任意类型的值
- 不论return语句出现在函数的什么位置，一旦得到执行将直接结束函数的执行。
- 如果函数没有return语句、有return语句但是没有执行到或者执行了不返回任何值的return语句，解释器都会认为该函数以return None结束，即返回空值。

- 斐波那契数列的函数并调用



- 嵌套函数的定义：

```
#嵌套函数定义1：
def func():
    x=1
    y=2
    m=3
    n=4
    def sum(a,b):
        return a+b
    def sub(a,b):
        return a-b
    return sum(x,y)*sub(m,n)
```

```
#嵌套函数定义2
def linear(a, b):
    def result(x):
        return a * x + b
    return result
```

2、函数的参数

- Python中任何东西都是对象，所以参数只支持引用传递的方式。
- Python通过名称绑定的机制，把实际参数的值和形式参数的名称绑定在一起，
  - 即把形式参数传递到函数所在的局部命名空间中，形式参数和实际参数指向内存中同一个存储空间。

- 对于绝大多数情况下，在函数内部直接修改形参的值不会影响实参，而是创建一个新变量。例如：

```
>>> def addOne(a):
    print(id(a), ':', a)
    a += 1
    print(id(a), ':', a)

>>> v = 3
>>> id(v)
1599055008
>>> addOne(v)
1599055008 : 3
1599055040 : 4
>>> v
3
>>> id(v)
1599055008
```

注意：此时a的地址与v的地址相同

现在a的地址和v的地址不一样了

- 在有些情况下，可以通过特殊的方式在函数内部修改实参的值。

```
>>> def modify(v):                    # 使用下标修改列表元素值
    v[0] = v[0]+1
>>> a = [2]                           #a为list
>>> modify(a)
>>> a
[3]
>>> def modify(v, item):              # 使用列表的方法为列表增加元素
    v.append(item)
>>> a = [2]
>>> modify(a,3)
>>> a
[2, 3]
```

- 也就是说，如果传递给函数的实参是可变序列，并且在函数内部使用下标或可变序列自身的方法增加、删除元素或修改元素时，实参也得到相应的修改。

```
>>> def modify(d):          #修改字典元素值或为字典增加元素
        d['age'] = 38
>>> a = {'name':'Dong', 'age':37, 'sex':'Male'}
>>> a
{'age': 37, 'name': 'Dong', 'sex': 'Male'}
>>> modify(a)
>>> a
{'age': 38, 'name': 'Dong', 'sex': 'Male'}
```

3. 默认值

- 函数的参数支持默认值。当某个参数没有传递实际的值时，函数将使用默认参数计算。
- 带默认值的参数不能位于没有默认值的参数前面

```
def greet(name,greeting='Hello'):
    print(greeting,name+'!')

greet('Bob')
greet('Bob','Good morning')
```

执行结果：

```
Hello Bob!
Good morning Bob!
```

4. 关键字参数

- 关键字参数有两大好处：
  - 清晰地指出了参数值，有助于提高程序的可读性；
  - 关键字参数的顺序无关紧要。
- 调用使用关键字参数的函数时，以**param=value**的方式传递参数

```
def shop(where ='store',what='pasta',howmuch='10 pounds'):
    print('I want you to go to the',where,'and buy',howmuch,'of', what+'.')

shop()
shop(what='towels')
shop(howmuch='a ton',what='towels')
shop(howmuch='a ton',what='towels',where='bakery')
```

执行结果：

```
I want you to go to the store and buy 10 pounds of pasta.
I want you to go to the store and buy 10 pounds of towels.
I want you to go to the store and buy a ton of towles.
I want you to go to the bakery and buy a ton of towles.
```

5. 可变长度参数

- 可变长度参数主要有两种形式：在参数名前加1个*或2个**
  - *parameter用来接受多个位置参数并将其放在一个元组中
  - **parameter接受多个关键参数并存放到字典中

```
>>> def demo(*p):
        print(p)


>>> demo(1,2,3)
(1, 2, 3)
>>> demo(1)
(1,)
```

```
>>> def demo(**p):
    for item in p.items():
        print(item)


>>> demo(x=1,y=2,z=3)
('y', 2)
('x', 1)
('z', 3)
```

- 几种不同类型的参数可以混合使用，但是不建议这样做。

```
>>> def func_4(a, b, c=4, *aa, **bb):
    print(a,b,c)
    print(aa)
    print(bb)

>>> func_4(1,2,3,4,5,6,7,8,9,xx='1',yy='2',zz=3)
(1, 2, 3)
(4, 5, 6, 7, 8, 9)
{'yy': '2', 'xx': '1', 'zz': 3}
>>> func_4(1,2,3,4,5,6,7,xx='1',yy='2',zz=3)
(1, 2, 3)
(4, 5, 6, 7)
{'yy': '2', 'xx': '1', 'zz': 3}
```

6、lambda表达式

- lambda表达式可以用来声明匿名函数，也就是没有函数名字的临时使用的小函数，
  - 尤其适合需要一个函数作为另一个函数参数的场合。也可以定义具名函数。
- lambda表达式只可以包含一个表达式，该表达式的计算结果可以看作是函数的返回值，不允许包含复合语句，但在表达式中可以调用其他函数。

```
>>> f = lambda x, y, z: x+y+z          #可以给lambda表达式起名字
>>> f(1,2,3)                           #像函数一样调用
6
>>> g = lambda x, y=2, z=3: x+y+z      #参数默认值
>>> g(1)
6
>>> g(2, z=4, y=5)                     #关键参数
11
>>> L = [(lambda x: x**2), (lambda x: x**3),(lambda x: x**4)]
>>> print(L[0](2),L[1](2),L[2](2))
4 8 16
>>> D = {'f1':(lambda:2+3), 'f2':(lambda:2*3),'f3':(lambda:2**3)}
>>> print(D['f1'](), D['f2'](), D['f3']())
5 6 8
```

7、Generator函数

- 包含yield语句的函数可以用来创建Generator对象，这样的函数也称Generator函数。其作用是作用是一次产生一个数据项
- yield语句与return语句的作用相似，都是用来从函数中返回值。与return语句不同的是：
  - return语句一旦执行会立刻结束函数的运行，
  - 而每次执行到yield语句并返回一个值之后会暂停或挂起后面代码的执行
  - 下次通过Generator对象的__next__()方法、内置函数next()、for循环遍历Generator对象元素或其他方式显式“索要”数据时恢复执行。
- Generator具有惰性求值的特点，适合大数据处理

例：用三种方法求Fibonacci数列的前N项

- 方法一：简单输出Fibonacci数列前 N 项

```
def fab(max):
    n, a, b = 0, 0, 1
    while n < max:
        print (b)
        a, b = b, a + b
        n = n + 1
```

执行结果：

```
>>> f=fab(5)
1
1
2
3
5
>>> type(f)
<class 'NoneType'>
```

- 方法二：定义一个函数，返回一个列表List，列表中包含了Fibonacci数列前 N 项

```
def fab(max):
    n, a, b = 0, 0, 1
    L = []
    while n < max:
        L.append(b)
        a, b = b, a + b
        n = n + 1
    return L
```

```
>>> f=fab(5)
>>> f
[1, 1, 2, 3, 5]
>>> type(f)
<class 'list'>
>>> for n in fab(5)
        print(n)


1
1
2
3
5
```

- 方法三：使用yield

```
def fab(max):
    n, a, b = 0, 0, 1
    while n < max:
        yield b
        # print b
        a, b = b, a + b
        n = n + 1
```

```
>>> f=fab(5)
>>> type(f)
<class 'generator'>
>>> f
<generator object fab at 0x000000C37F60D8E0>
>>> for n in fab(5):
        print(n)


1
1
2
3
5
```

说明：

- 方法三与方法一相比，仅仅把 print b 改为了 yield b，就在保持简洁性的同时获得了 iterable 的效果。
- 也可以手动调用 fab(5) 的 __next__() 方法，这样可以更清楚地看到 fab 的执行流程。
- 当函数执行结束时，generator 自动抛出 StopIteration 异常，表示迭代完成。在 for 循环里，无需处理 StopIteration 异常，循环会正常结束。

```
>>> f=fab(5)
>>> f.__next__()
1
>>> f.__next__()
1
>>> f.__next__()
2
>>> f.__next__()
3
>>> f.__next__()
5
>>> f.__next__()
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    f.__next__()
StopIteration
```

本节主要内容

- 概述
- Python安装和使用
- Python的编码规范
- 变量和常量
- 数据类型
- 运算符与表达式
- 流程控制
- 函数
- **Python常用内置函数**
- 文件处理
- 目录的常见操作
- 面向对象编程
- 异常
- 数据库编程

Python常用内置函数

- 内置函数（BIF，built-in functions）是Python内置对象类型之一，不需要额外导入任何模块即可直接使用
- 这些内置对象都封装在内置模块__builtins__之中，用C语言实现并且进行了大量优化，具有非常快的运行速度。
- 使用内置函数dir()可以查看所有内置函数和内置对象：

```
>>> dir(__builtins__)
```

- 使用help(函数名)可以查看某个函数的用法。

```
>>> help(sum)
```

函数	功能简要说明
abs(x)	返回数字x的绝对值或复数x的模
all(iterable)	如果对于可迭代对象中所有元素x都等价于True，也就是对于所有元素x都有bool(x)等于True，则返回True。对于空的可迭代对象也返回True
any(iterable)	只要可迭代对象iterable中存在元素x使得bool(x)为True，则返回True。对于空的可迭代对象，返回False
ascii(obj)	把对象转换为ASCII码表示形式，必要的时候使用转义字符来表示特定的字符
bin(x)	把整数x转换为二进制串表示形式
bool(x)	返回与x等价的布尔值True或False
bytes(x)	生成字节串，或把指定对象x转换为字节串表示形式
callable(obj)	测试对象obj是否可调用。类和函数是可调用的，包含__call__()方法的类的对象也是可调用的
compile()	用于把Python代码编译成可被exec()或eval()函数执行的代码对象
complex(real, [imag])	返回复数
chr(x)	返回Unicode编码为x的字符

函数	功能简要说明
delattr(obj, name)	删除属性，等价于del obj.name
dir(obj)	返回指定对象或模块obj的成员列表，如果不带参数则返回当前作用域内所有标识符
divmod(x, y)	返回包含整商和余数的元组((x-x%y)/y, x%y)
enumerate(iterable[, start])	返回包含元素形式为(0, iterable[0]), (1, iterable[1]), (2, iterable[2]), ...的迭代器对象
eval(s[, globals[, locals]])	计算并返回字符串s中表达式的值
exec(x)	执行代码或代码对象x
exit()	退出当前解释器环境
filter(func, seq)	返回filter对象，其中包含序列seq中使得单参数函数func返回值为True的那些元素，如果函数func为None则返回包含seq中等价于True的元素的filter对象
float(x)	把整数或字符串x转换为浮点数并返回
frozenset([x])	创建不可变的集合对象
getattr(obj, name[, default])	获取对象中指定属性的值，等价于obj.name，如果不存在指定属性则返回default的值，如果要访问的属性不存在并且没有指定default则抛出异常

函数	功能简要说明
globals()	返回包含当前作用域内全局变量及其值的字典
hasattr(obj, name)	测试对象obj是否具有名为name的成员
hash(x)	返回对象x的哈希值，如果x不可哈希则抛出异常
help(obj)	返回对象obj的帮助信息
hex(x)	把整数x转换为十六进制串
id(obj)	返回对象obj的标识（内存地址）
input([提示])	显示提示，接收键盘输入的内容，返回字符串
int(x[, d])	返回实数（float）、分数（Fraction）或高精度实数（Decimal）x的整数部分，或把d进制的字符串x转换为十进制并返回，d默认为十进制
isinstance(obj, class-or-type-or-tuple)	测试对象obj是否属于指定类型（如果有多个类型的话需要放到元组中）的实例
iter(...)	返回指定对象的可迭代对象
len(obj)	返回对象obj包含的元素个数，适用于列表、元组、集合、字典、字符串以及range对象和其他可迭代对象

函数	功能简要说明
list([x])、set([x])、tuple([x])、dict([x])	把对象x转换为列表、集合、元组或字典并返回，或生成空列表、空集合、空元组、空字典
locals()	返回包含当前作用域内局部变量及其值的字典
map(func, *iterables)	返回包含若干函数值的map对象，函数func的参数分别来自于iterables指定的每个迭代对象，
max(x)、 min(x)	返回可迭代对象x中的最大值、最小值，要求x中的所有元素之间可比较大小，允许指定排序规则和x为空时返回的默认值
next(iterator[, default])	返回可迭代对象x中的下一个元素，允许指定迭代结束之后继续迭代时返回的默认值
oct(x)	把整数x转换为八进制串
open(name[, mode])	以指定模式mode打开文件name并返回文件对象
ord(x)	返回1个字符x的Unicode编码
pow(x, y, z=None)	返回x的y次方，等价于x ** y或(x ** y) % z

函数	功能简要说明
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)	基本输出函数
quit()	退出当前解释器环境
range([start,] end [, step])	返回range对象，其中包含左闭右开区间[start,end)内以step为步长的整数
reduce(func, sequence[, initial])	将双参数的函数func以迭代的方式从左到右依次应用至序列seq中每个元素，最终返回单个值作为结果。在Python 2.x中该函数为内置函数，在Python 3.x中需要从functools中导入reduce函数再使用
repr(obj)	返回对象obj的规范化字符串表示形式，对于大多数对象有eval(repr(obj))==obj
reversed(seq)	返回seq（可以是列表、元组、字符串、range以及其他可迭代对象）中所有元素逆序后的迭代器对象

函数	功能简要说明
round(x [, 小数位数])	对x进行四舍五入，若不指定小数位数，则返回整数
sorted(iterable, key=None, reverse=False)	返回排序后的列表，其中iterable表示要排序的序列或迭代对象key用来指定排序规则或依据，reverse用来指定升序或降序。该函数不改变iterable内任何元素的顺序
str(obj)	把对象obj直接转换为字符串
sum(x, start=0)	返回序列x中所有元素之和，返回start+sum(x)
type(obj)	返回对象obj的类型
zip(seq1 [, seq2 [...]])	返回zip对象，其中元素为(seq1[i], seq2[i], ...)形式的元组最终结果中包含的元素个数取决于所有参数序列或可迭代对象中最短的那个

类型转换与类型判断

- **list()、tuple()、dict()、set()、frozenset()用来把其他类型的数据转换成为列表、元组、字典、可变集合和不可变集合，或者创建空列表、空元组、空字典和空集合。**

```
>>> list(range(5))                    #把range对象转换为列表
[0, 1, 2, 3, 4]
>>> tuple(_)                          #一个下划线表示上一次正确的输出结果
(0, 1, 2, 3, 4)
>>> dict(zip('1234', 'abcde'))        #创建字典
{'4': 'd', '2': 'b', '3': 'c', '1': 'a'}
>>> set('1112234')                    #创建可变集合，自动去除重复
{'4', '2', '3', '1'}
>>> _.add('5')
>>> _
{'2', '1', '3', '4', '5'}
>>> frozenset('1112234')              #创建不可变集合，自动去除重复
frozenset({'2', '1', '3', '4'})
>>> _.add('5')                        #不可变集合frozenset不支持元素添加与删除
AttributeError: 'frozenset' object has no attribute 'add'
```

类型转换与类型判断

- 内置函数type()和isinstance()可以用来判断数据类型，常用来对函数参数进行检查，可以避免错误的参数类型导致函数崩溃或返回意料之外的结果。

```
>>> type([3])                                  #查看[3]的类型
<class 'list'>
>>> type({3}) in (list, tuple, dict)           #判断{3}是否为list,tuple或dict类型的实例
False
>>> type({3}) in (list, tuple, dict, set)      #判断{3}是否为list,tuple,dict或set的实例
True
>>> isinstance(3, int)                         #判断3是否为int类型的实例
True
>>> isinstance(3j, int)
False
>>> isinstance(3j, (int, float, complex))      #判断3是否为int,float或complex类型
True
```

最值与求和

- max()、min()、sum()这三个内置函数分别用于计算列表、元组或其他包含有限个元素的可迭代对象中所有元素最大值、最小值以及所有元素之和。
- sum()默认（可以通过start参数来改变）支持包含数值型元素的序列或可迭代对象，max()和min()则要求序列或可迭代对象中的元素之间可比较大小

```
>>> from random import randint
>>> a = [randint(1,100) for i in range(10)]   #包含10个[1,100]之间随机数的列表
>>> print(max(a), min(a), sum(a))             #最大值、最小值、所有元素之和
>>> sum(a) / len(a)                           #平均值
```

基本输入输出

- input()和print()是Python的基本输入输出函数：
  - Input()：用来接收用户的键盘输入
  - Print()：用来把数据以指定的格式输出到标准控制台或指定的文件对象。
- 不论用户输入什么内容，input()一律返回字符串，必要的时候可以使用内置函数int()、float()或eval()对用户输入的内容进行类型转换。

基本输入输出

```
>>> x = input('Please input: ')
Please input: 345
>>> x
'345'
>>> type(x)                          #把用户的输入作为字符串对待
<class 'str'>
>>> int(x)                           #转换为整数
345
>>> eval(x)                          #对字符串求值，或类型转换
345
>>> x = input('Please input: ')
Please input: [1, 2, 3]
>>> x
'[1, 2, 3]'
>>> type(x)
<class 'str'>
>>> eval(x)
[1, 2, 3]
```

基本输入输出

- 函数print()用于输出信息到标准控制台或指定文件，语法格式为：

```
print(value1, value2, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

  - sep参数之前为需要输出的内容（可以有多个）；
  - sep参数用于指定数据之间的分隔符，默认为空格；
  - end参数用于指定输出完数据之后再输出什么字符；
  - file参数用于指定输出位置，默认为标准控制台，也可以重定向输出到文件。

```
>>> print(1, 3, 5, 7, sep='\t')          #修改默认分隔符
1       3       5       7
>>> for i in range(10):                  #修改end参数，每个输出之后不换行
    print(i, end=' ')
0 1 2 3 4 5 6 7 8 9
>>> with open('test.txt', 'a+') as fp:
    print('Hello world!', file=fp)       #重定向，将内容输出到文件中
```

range()

- range() 语法格式为range([start,] end [, step])，有三种用法：
  - range(stop)
  - range(start, stop)
  - range(start, stop, step)
- 该函数返回具有惰性求值特点的range对象，其中包含左闭右开区间[start,end)内以step为步长的整数。参数start默认为0，step默认为1。

range()

```
>>> range(5)                          #start默认为0，step默认为1
range(0, 5)
>>> list(_)
[0, 1, 2, 3, 4]
>>> list(range(1, 10, 2))             #指定起始值和步长
[1, 3, 5, 7, 9]
>>> list(range(9, 0, -2))             #步长为负数时，start应比end大
[9, 7, 5, 3, 1]
```

技术不会停下脚步，学习永无止境。