

山东科技大学——测绘与空间信息学院

Python程序设计

地理信息科学系 刘洪强

J6-557 电话：86081170

2021年3月3日星期三

课程安排:

36个学时，其中授课24个学时，实验12个学时

章节内容

第1章 认识Python

第2章 Python编程基础

第3章 函数、类、包和模块

第4章 文件操作

第5章 地图文档管理与数据访问

第6章 空间数据定位与查询

第7章 空间数据分析

第8章 地图制图与输出

第2章 Python编程基础

主要内容

变量

运算符和表达式

数据类型

输入和输出语句

赋值和注释语句

条件语句

循环语句

第2章 Python编程基础

2.1 变量

在Python中，**变量**是由数字或字符组成的任意长度的字符串，但必须以字母或下划线开头，python是区分大小写的。

在Python中，等号（=）是赋值语句，使用环境非常宽松，可以把**任意数据类型**赋值给变量,而且可以随时**改变为其它类型的值**。

如下为定义一个名为xiaoMing的变量：

```
>>> xiaoMing='XiaoMing'
```

```
>>> xiaoMing=123
```

第2章 Python编程基础

2.2 运算符和表达式

运算符是构成语句的连接符，包括加（+）、减（-）、乘（*）、除（/）、地板除（//）、取余（%）等都是运算符，是一些特殊符号的集合。操作对象就是由运算符连接起来的对象。

Python支持以下8种运算符：

- （1）算术运算符。 （2）比较（关系）运算符。
- （3）赋值运算符。 （4）逻辑运算符。
- （5）位运算符。 （6）成员运算符。
- （7）身份运算符。 （8）运算符优先级。

第2章 Python编程基础

2.2 运算符和表达式

(1) 算术运算符

a = 10 b=20

运算符	描述	实例
+	加：两个对象相加	a + b 输出结果 30
-	减：得到负数或是一个数减去另一个数	a - b 输出结果 -10
*	乘：两个数相乘或是返回一个被重复若干次的字符串	a * b 输出结果 200
/	除：x除以y	a / b 输出结果 0.5
%	取模：返回除法的余数	b % a 输出结果 0
**	幂：返回x的y次幂	a**b 为10的20次方， 输出结果 100000000000000000000
//	取整除（地板除）：返回商的整数部分（向下取整）	9//2 输出结果 4 -9//2 输出结果 -5

注意：Python2.X里，整数除整数，只能得出整数。
如果要得到小数部分，把其中一个数改成浮点数即可。

第2章 Python编程基础

2.2 运算符和表达式

(2) 比较运算符

a = 10 b=5

运算符	描述	实例
==	等于：比较对象是否相等	(a == b) 返回 False。
!=	不等于：比较两个对象是否不相等	(a != b) 返回 True.
>	大于：返回x是否大于y	(a > b) 返回 True。
<	小于：返回x是否小于y。	(a < b) 返回 False。
>=	大于等于：返回x是否大于等于y。	(a >= b) 返回 True。
<=	小于等于：返回x是否小于等于y。	(a <= b) 返回 False。

第2章 Python编程基础

2.2 运算符和表达式

(3) 赋值运算符

运算符	描述	实例
<code>--</code>	减法赋值运算符	<code>c -= a</code> 等效于 <code>c = c - a</code>
<code>*=</code>	乘法赋值运算符	<code>c *= a</code> 等效于 <code>c = c * a</code>
<code>/=</code>	除法赋值运算符	<code>c /= a</code> 等效于 <code>c = c / a</code>
<code>%=</code>	取模赋值运算符	<code>c %= a</code> 等效于 <code>c = c % a</code>
<code>**=</code>	幂赋值运算符	<code>c **= a</code> 等效于 <code>c = c ** a</code>
<code>//=</code>	取整（地板）除赋值运算符	<code>c //= a</code> 等效于 <code>c = c // a</code>

第2章 Python编程基础

2.2 运算符和表达式

(4) 位运算符

```
a = 0011 1100
b = 0000 1101
-----
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a = 1100 0011
```

运算符	描述	实例
&	按位与运算符	(a & b) 输出结果 12 ， 二进制解释： 0000 1100
	按位或运算符	(a b) 输出结果 61 ， 二进制解释： 0011 1101
^	按位异或运算符：	(a ^ b) 输出结果 49 ， 二进制解释： 0011 0001
~	按位取反运算符：	(~a) 输出结果 -61 ， 二进制解释： 1100 0011， 在一个有符号二进制数的补码形式。
<<	左移动运算符： <<“右边的数指定移动的位数，高位丢弃，低位补0。	a << 2 输出结果 240 ， 二进制解释： 1111 0000
>>	右移动运算符：	a >> 2 输出结果 15 ， 二进制解释： 0000 1111

第2章 Python编程基础

2.2 运算符和表达式

(5) 逻辑运算符

a = 10

b = 20

运算符	逻辑表达式	描述	实例
And	x and y	布尔"与" - 如果 x 为 False，x and y 返回 False，否则它返回 y 的计算值。	(a and b) 返回 20。
Or	x or y	布尔“或” - 如果 x 是非 0，它返回 x 的值，否则它返回 y 的计算值。	(a or b) 返回 10。
Not	not x	布尔“非” - 如果 x 为 True，返回 False。如果 x 为 False，它返回 True。	not(a and b)返回 False

第2章 Python编程基础

2.2 运算符和表达式

(6) 成员运算符

运算符	描述	实例
in	如果在指定的序列中找到值返回 True ，否则返回 False 。	如果x在y序列中返回 True 。
not in	如果在指定的序列中没有找到值返回 True ，否则返回 False 。	如果x不在y序列中返回 True 。

```
a = 10
list = [1, 2, 3, 4, 5]
if ( a in list ):
    print "变量 a 在给定的列表中 list 中"
else:
    print "变量 a 不在给定的列表中 list 中"
```

第2章 Python编程基础

2.2 运算符和表达式

(7) 身份运算符

身份运算符用于比较两个对象的存储单元

运算符	描述	实例
is	is判断两个标识符是不是引用自一个对象	x is y, 类似 id(x) == id(y) , 如果引用的是同一个对象则返回 True, 否则返回 False
is not	is not用于判断两个标识符是不是引用自不同对象	x is not y , 类似 id(a) != id(b)。如果引用的不是同一个对象则返回结果 True, 否则返回 False。

```
a = 20
b = 20
if ( a is b ):
    print("a 和 b 有相同的标识")
else:
    print("a 和 b 没有相同的标识")
```

第2章 Python编程基础

2.2 运算符和表达式 (8) 运算符优先级

运算符	描述
**	指数 (最高优先级)
~ + -	按位翻转, 一元加号和减号 (最后两个为 +@ 和 -@)
* / % //	乘, 除, 取模和取整除
+ -	加法减法
>> <<	右移, 左移运算符
&	位 'AND'
^	位运算符
<= < > >=	比较运算符
<> == !=	等于运算符
= %= /= //= -= += *= **=	赋值运算符
is is not	身份运算符
in not in	成员运算符
not or and	逻辑运算符

第2章 Python编程基础

2.2 运算符和表达式

(9) 表达式

表达式是值、变量和操作符的组合。单独一个值也被看作一个表达式。单独的变量也可以看作一个表达式。

表达式和语句一般不容易区分，很多人会将两者混在一起。那么语句和表达式之间有什么区别？

表达式是某事，而**语句**是做某事，说的通俗点就是告诉计算机做什么。比如 $3*3$ 是9，而`print(3*3)`打印出来是9。那么区别在哪里？

比如我们在交互模式下输入如下：

```
>>> 3*3
```

```
9
```

```
>>> print(3*3)
```

```
9
```


第2章 Python编程基础

2.3 数据类型——数字类型

五种基本数字类型：

int 1 2 0x80 -0xA9

long 12345678902010 -0xABCDEF123456

bool True False

float 3.1415926 -1.2E-14 10.32.1e10

Complex 6.54+3.21j -1.23+45.6J 0+1j 99-88j -0.142857+0j

(**注：**long 类型只存在于 Python2.X 版本中，在 2.2 以后的版本中，int 类型数据溢出后会自动转为long类型。在 Python3.X 版本中 long 类型被移除，使用 int 替代。)

第2章 Python编程基础

2.3 数据类型——数字类型

(1) 整型

整型(int)，通常被称为是整型或整数，是正或负整数，**不带小数点**。

例如交互模式下输入如下：

```
>>> 51
```

```
51
```

这里使用的就是整型。

整型加法如下：

```
>>> 25+25
```

```
50
```

第2章 Python编程基础

2.3 数据类型——数字类型

(1) 整型

整型减法：

```
>>> 51-50
```

```
1
```

整型乘法：

```
>>> 51*2
```

```
102
```

整型除法：

```
>>> 153/51
```

```
3.0
```

```
>>> 155/51
```

```
3.0392156862745097
```

此处出现除不尽的情况了。

第2章 Python编程基础

2.3 数据类型——数字类型

(1) 整型

在整数除法中，**除法** (/) 计算结果是浮点数，即使是两个整数恰好整除，结果也是**浮点数**，如果只想得到整数的结果，丢弃可能的分数部分，可以使用**地板除** (//)，整数的地板除 (//) 永远是**整数**，即使除不尽。

改成如下写法：

```
>>> 153//51
```

```
3
```

```
>>> 155//51
```

```
3
```

Python还提供一个**余数运算**，可以得到两个整数相除的余数。如下：

```
>>> 153%51
```

```
0
```

```
>>> 155%51
```

```
2
```

第2章 Python编程基础

2.3 数据类型——数字类型

(2) 浮点型

浮点型(float)，浮点型由整数部分与小数部分组成，浮点型也可以使用科学计数法表示。

先看示例：

```
>>> 3.3*102
```

```
336.59999999999997
```

按预计应该一位小数，但输出结果却有这么多位小数。是因为整数和浮点数在计算机内部存储的方式是不同的，整数运算永远是精确的，而浮点数运算则可能会有四舍五入的误差。

第2章 Python编程基础

2.3 数据类型——数字类型

(2) 浮点型

浮点运算：

```
>>> 3.3*102+15.5  
352.09999999999997
```

浮点除法：

```
>>> 153/51.0  
3.0
```

```
>>> 155/51.0  
3.0392156862745097
```

浮点地板除：

```
>>> 155//51.0  
3.0
```

```
>>> 155%51.0  
2.0
```

第2章 Python编程基础

2.3 数据类型——数字类型

(3) 复数

复数 (complex)，复数由实数部分和虚数部分构成，可以用 $a + bj$ ，或者 `complex(a, b)` 表示，复数的实部 a 和虚部 b 都是浮点型。

Python支持复数，常用的运算包括：

复数可以用使用函数 `complex(real, imag)` 或者是带有后缀 j 的浮点数来指定。比如：

```
>>> a = complex(2, 4)
```

```
>>> b = 3 - 5j
```

```
>>> a
```

```
(2+4j)
```

```
>>> b
```

```
(3-5j)
```

例：Exp2_2 Complex

第2章 Python编程基础

2.3 数据类型——数字类型

(3) 复数

获取复数的实部、虚部和共轭复数，比如：

```
>>> a.real
2.0
>>> a.imag
4.0
>>> a.conjugate()
(2-4j) 29+0.6470588235294118j
>>> abs(a)
4.47213595499958
```


第2章 Python编程基础

2.3 数据类型——数字类型

(3) 复数

常见的数学运算：

```
>>> a + b
```

```
(5-1j)
```

```
>>> a * b
```

```
(26+2j)
```

```
>>> a / b
```

```
(-0.4117647058823529+0.6470588235294118j)
```

```
>>> abs(a)
```

```
4.47213595499958
```

第2章 Python编程基础

2.3 数据类型——数字类型

(3) 复数

其他的复数函数比如正弦、余弦或平方根，使用 `cmath` 模块：

```
>>> import cmath
>>> cmath.sin(a)
(24.83130584894638-11.356612711218174j)
>>> cmath.cos(a)
(-11.36423470640106-24.814651485634187j)
>>> cmath.exp(a)
(-4.829809383269385-5.5920560936409816j)
```

第2章 Python编程基础

2.3 数据类型——数字类型

(4) 常的数学函数

`abs(x)` 求数值的绝对值

`ceil(x)` 取顶

`divmod(x)` 返回两个数值的商和余数

`floor(x)` 取底

`fabs(x)` 取绝对值

`factorial(x)` 阶乘

`hypot(x, y)` 计算 $\sqrt{x^2+y^2}$

`pow(x, y)` x 的 y 次方

`sqrt(x)` 开平方

`log(x)/log10(x)`

`max(x)` 返回最大值

`Min(x)` 返回最小值

`round(x)` 返回浮点数进行四舍五入的值

`sum(x)` 返回和

`trunc(x)` 截断取整数部分

`isnan(x)` 判断是否NaN

`degrees(x)` 弧度转角度

`radians(x)` 角度转弧度

`sin(x)`

`cos(x)`

`tan(x)`

`asin(x)`

`acos(x)`

`atan(x)`

第2章 Python编程基础

2.3 数据类型——字符串

字符串： []索引操作符 [:]切片操作符 +连接运算 *重复运算

```
>>> strPython = "Python"
>>> strIsCool = "is cool!"
>>> strPython[0]
"P"    # 第1个字符
>>> strPython[2:5]
"tho"  # 第3到第5个字符
>>> strIsCool[-1]
"!"    # 最后一个字符
>>> strIsCool[:2]
"is"   # 第1到第2个字符
>>> strPython + "\n" + strIsCool    # \n: 回车
"Python is cool!"
>>> "-" * 30
"-----"
```

第2章 Python编程基础

2.3 数据类型——字符串

字符串必须以引号标记开始，并以引号标记结束；可以是单引号、双引号和三单引号或三双引号。

- (1) 单引号中可以使用双引号，中间的会当作字符串输出
- (2) 双引号中可以使用单引号，中间的会当作字符串输出
- (3) 三单引号和三双引号中间的字符串在输出时保持原来的格式。

第2章 Python编程基础

2.3 数据类型——字符串

字符串操作的常用函数：

(1) 字母处理函数

```
.upper()      # 全部大写  
.lower()      # 全部小写  
.swapcase()   # 大小写互换  
.capitalize() # 首字母大写，其余小写  
.title()      # 首字母大写
```

第2章 Python编程基础

2.3 数据类型——字符串

字符串操作的常用函数：

(2) 格式化相关函数

- .ljust(width) # 获取固定长度，左对齐，右边不够用空格补齐
- .rjust(width) # 获取固定长度，右对齐，左边不够用空格补齐
- .center(width) # 获取固定长度，中间对齐，两边不够用空格补齐
- .zfill(width) # 获取固定长度，右对齐，左边不足用0补齐

第2章 Python编程基础

2.3 数据类型——字符串

字符串操作的常用函数：

(3) 字符串搜索相关函数

- .find() # 搜索指定字符串，没有返回-1
- .index() # 同上，但是找不到会报错
- .rfind() # 从右边开始查找
- .count() # 统计指定的字符串出现的次数

第2章 Python编程基础

2.3 数据类型——字符串

字符串操作的常用函数：

(3) 字符串替换函数

```
.replace('old', 'new')      # 替换old为new  
.replace('old', 'new', 次数)  # 替换指定次数的old为new
```

例：

```
s='hello world'  
print(s.replace('world', 'python'))  
print(s.replace('l', 'p', 2))  
print(s.replace('l', 'p', 5))  
hello python  
heppo world  
heppo worpd
```

第2章 Python编程基础

2.3 数据类型——字符串

字符串操作的常用函数：

(4) 去空格及去指定字符函数

`.strip()` # 去两边空格

`.lstrip()` # 去左边空格

`.rstrip()` # 去右边空格

`.split()` # 默认按空格分隔

`.split('指定字符')` # 按指定字符分割字符串为数组

例：

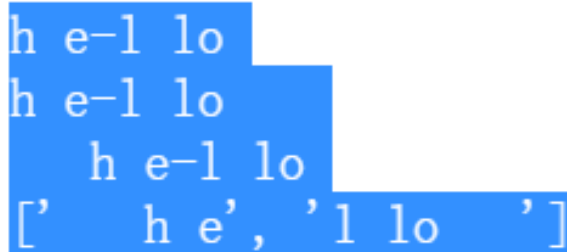
```
s='  h e-l lo  '
```

```
print(s.strip())
```

```
print(s.lstrip())
```

```
print(s.rstrip())
```

```
print(s.split('-'))
```



```
h e-l lo  
h e-l lo  
h e-l lo  
[' h e', 'l lo ']
```

第2章 Python编程基础

2.3 数据类型——字符串

字符串操作的常用函数：

(5) 字符串判断相关函数

```
.startswith('start')    # 是否以start开头  
.endswith('end')        # 是否以end结尾  
.isalnum()              # 是否全为字母或数字  
.isalpha()              # 是否全字母  
.isdigit()              # 是否全数字  
.islower()              # 是否全小写  
.isupper()              # 是否全大写  
.istitle()              # 判断首字母是否为大写  
.isspace()              # 判断字符是否为空格
```

第2章 Python编程基础

2.3 数据类型——字符串

字符串操作的常用函数：

(6) 其它相关函数

- .bin() # 十进制数转八进制
- .hex() # 十进制数转十六进制
- .range() # 函数：可以生成一个整数序列
- .type() # 查看数据类型
- .len() # 计算字符串长度
- .format() # 格式化字符串，类似%s，传递值能多不能少

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(1) 序列

在Python中，最基本的数据结构是**序列**（sequence）。

Python包含6种内建的序列，即**列表**、**元组**、**字符串**、**Unicode字符串**、**buffer对象**和**xrange对象**。

通用序列操作包括：

索引（indexing）、

分片（slicing）、

序列相加（adding）、

乘法（multiplying）、

成员资格、

长度、

最小值和最大值。

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(1) 序列

1) 索引 (indexing)

序列中的每个元素都分配一个数字，代表它在序列中的位置，或索引，第一个索引是0，第二个索引是1，依此类推。

序列中所有的元素都是有编号的——从0开始递增。可以通过编号分别对序列的元素进行访问。

例如：

```
>>> greeting='Hello'
```

```
>>> greeting[0]
```

```
'H'
```

```
>>> greeting[1]
```

```
'e'
```

```
>>> greeting[2]
```

```
'l'
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(1) 序列

1) 索引 (indexing)

Python的序列也可以从右边开始索引，最右边的一个元素的索引为-1，向左开始递减。

在Python中，从左向右索引称为正数索引，从右向左称为负数索引。使用负数索引时，Python会从最后1个元素开始计数。最后一个元素的位置编号是-1。

例如：

```
>>> greeting[-1]
```

```
'o'
```

```
>>> greeting[-2]
```

```
'l'
```

```
>>> greeting[-3]
```

```
'l'
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(1) 序列

2) 分片 (slicing)

使用分片可以对一定范围内的元素进行访问，分片通过冒号相隔的两个索引来实现，也可以设置步长。

分片操作既支持正数索引，也支持负数索引，并且分片操作对于提取序列的一部分是很方便的。

分片操作的实现需要提供两个索引作为边界，若两个索引之间没有元素，则返回空序列。

例如：

```
>>> number=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> number[-3:-1]
```

```
[8, 9]
```

```
>>> number[-3:0]
```

```
[]
```


第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(1) 序列

2) 分片 (slicing)

对于一个**正数步长**，Python会从序列的头部开始向右提取元素，直到最后一个元素；正数步长，必须让开始点小于结束点；而对于**负数步长**，则是从序列的尾部开始向左提取元素，直到第一个元素；而负数步长，则必须让开始点大于结束点。

例如：

```
>>> number=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> number[0:10:3]
```

```
[1, 4, 7, 10]
```

```
>>> number[10:0:-3]
```

```
[10, 7, 4, 1]
```

```
>>> number[-1:-10:-3]
```

```
[10, 7, 4]
```

```
>>> number[-10:-1:3]
```

```
[1, 4, 7, 10]
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(1) 序列

3) 序列相加

通过使用加号可以进行序列的连接操作，**只有类型相同**的序列才能通过加号进行序列连接操作，输入如下：

例如：

```
>>> [1, 2, 3]+[4, 5, 6]
```

```
[1, 2, 3, 4, 5, 6]
```

```
>>> a=[1, 2]
```

```
>>> b=[5, 6]
```

```
>>> a+b
```

```
[1, 2, 5, 6]
```

```
>>> s='hello, '
```

```
>>> w='world'
```

```
>>> s+w
```

```
'hello, world'
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(1) 序列

3) 序列相乘

用一个数字x乘以一个序列会生成新的序列，在新的序列中，原来的序列将被**重复x次**，这个就是序列中的乘法。输入如下：

例如：

```
>>> 'hello'*5
```

```
'hellohellohellohellohello'
```

```
>>> [7]*10
```

```
[7, 7, 7, 7, 7, 7, 7, 7, 7, 7]
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(1) 序列

4) 成员资格

in运算符用于检验某个条件是否为真，检查一个值是否在序列中，并返回检验结果，检验结果为真返回True，结果为假则返回False。

例如：

```
>>> greeting='hello, world'
>>> 'w' in greeting    #检测w是否在字符串中
True
>>> 'a' in greeting
False
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(1) 序列

5) 长度、最小值和最大值

Python为我们提供了长度、最大值和最小值的内建函数，对应的内建函数分别为`len`、`max`和`min`。`max`和`min`函数的参数也可以是个数字。

例如：

```
>>> numbers=[300, 200, 100, 800, 500]
```

```
>>> len(numbers)
```

```
5
```

```
>>> max(numbers)
```

```
800
```

```
>>> min(numbers)
```

```
100
```

```
>>> max(5, 3, 10, 7)
```

```
10
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(2) 列表

列表(list)和元组(tuple)，是序列的特殊类型，可以看成普通的“数组”。

列表是可变的(mutable)，可以动态改为元素的值。

1) 元素赋值

```
a=[1, 2, 3, 2, 1]
```

```
>>> a[1]=10
```

```
>>> a
```

```
[1, 10, 3, 2, 1]
```

```
>>> a[3]=10
```

```
>>> a
```

```
[1, 10, 3, 10, 1]
```

注：不能为一个不存在元素的位置赋值，如：

```
a[6]=10
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(2) 列表

2) 增加元素

`append()` 方法是一个用于在列表末尾添加新的对象的方法。该方法的语法如下：

```
list.append(obj)
```

此语法中`list`代表的是列表，`obj`代表的是需要添加到`list`列表末尾的对象。

例如：

```
>>> tring=[1,2,3]
>>> tring.append(4)
>>> tring
[1, 2, 3, 4]
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(2) 列表

3) 删除元素

可以使用del删除列表中的元素。

该方法的语法如下：

```
Del list[index]
```

此语法中list代表的是列表，index代表的索引号，可是连续的索引号。

例如：

```
>>> tring=['a','b','c','d','e']
```

```
>>> len(tring)
```

```
5
```

```
>>> del tring[1]
```

```
>>> len(tring)
```

```
4
```

```
>>> del tring[1:3]
```


第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(2) 列表

4) 分片赋值

可以对列表进行分片赋值例如：

```
>>> boil=list('女排夺冠了')
>>> boil
['女', '排', '夺', '冠', '了']
>>> tring[3:5]=[1,45,'last']
>>> tring
['a', 'e', 1, 45, 'last']
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(2) 列表

5) 嵌套列表

在列表中可以嵌套列表，在列表中嵌套的列表取出后还是列表。

```
>>> field=['a','b','c']
```

```
>>> num=[1,2,3]
```

```
>>> mix=[field,num]
```

```
>>> mix
```

```
 [['a', 'b', 'c'], [1, 2, 3]]
```

```
>>> mix[0]
```

```
 ['a', 'b', 'c']
```

```
>>> mix[1]
```

```
 [1, 2, 3]
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(2) 列表

6) 列表的方法

序号	函数	描述
1	<code>list.append(obj)</code>	在列表的末尾添加新的对象。
2	<code>list.count(obj)</code>	统计某个元素在列表中出现的次数。
3	<code>list.extend(seq)</code>	在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）。
4	<code>list.index(obj)</code>	从列表中找出某个值第一个匹配项的索引位置。
5	<code>list.insert(index,obj)</code>	在列表指定位置插入元素。
6	<code>list.pop(obj=list[-1])</code>	移除列表中的一个元素（默认最后一个元素），并且返回该元素的值。

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(2) 列表

6) 列表的方法

序号	函数	描述
7	<code>list.remove(obj)</code>	移除列表中某个值的第一个匹配项。
8	<code>list.reverse()</code>	反向列表中元素。
9	<code>list.sort(func)</code>	对原列表进行排序，如果指定参数，则使用参数指定的比较方法进行排序。
10	<code>list.clear()</code>	清空列表，类似于 <code>del a[:]</code> 。
11	<code>list.copy()</code>	复制列表。

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(2) 列表

6) 列表的方法

1.append() 向列表尾部追加一个新元素，列表只占一个索引位，在原有列表上增加。

2.extend() 向列表尾部追加一个列表，将列表中的每个元素都追加进来，在原有列表上增加。

3.+ 直接用+号看上去与用extend()一样的效果，但是实际上是生成了一个新的列表存这两个列表的和，只能用在两个列表相加上。

4.+= 效果与extend()一样，向原列表追加一个新元素，在原有列表上增加。

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(2) 列表

6) 列表的方法

```
>>> A=[1,2,3,4]
>>> B=[5,6,7,8]
>>> A.append(B)
>>> A
[1, 2, 3, 4, [5, 6, 7, 8]]
>>> A=[1,2,3,4]
>>> B=[5,6,7,8]
>>> A.extend(B)
>>> A
[1, 2, 3, 4, 5, 6, 7, 8]
>>> A=[1,2,3,4]
>>> B=[5,6,7,8]
>>> A=A+B
>>> A
[1, 2, 3, 4, 5, 6, 7, 8]
>>> A=[1,2,3,4]
>>> B=[5,6,7,8]
>>> A+=B
>>> A
[1, 2, 3, 4, 5, 6, 7, 8]
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(2) 列表

6) 列表的方法

sort方法有两个可选参数——**key**和**reverse**。例如：

```
>>> field=['study','python','is','happy']
```

```
>>> field.sort(key=len) #按字符串由短到长排序
```

```
>>> field
```

```
['is', 'study', 'happy', 'python']
```

```
>>> field.sort(key=len,reverse=True) #按字符串由长到短排序，  
传递两个参数
```

```
>>> field
```

```
['python', 'study', 'happy', 'is']
```

```
>>> num=[5,8,1,3,6]
```

```
>>> num.sort(reverse=True) #排序后逆序
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(3) 元组

Python的元组与列表类似，不同之处在于元组的元素不能修改，使用逗号分隔了一些值，那么你就自动创建了元组。

```
>>> 1,2,3
```

```
(1, 2, 3)
```

```
>>> 'hello','world'
```

```
('hello', 'world')
```


第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(3) 元组

tuple函数的功能和**list**函数基本上是一样的：以一个序列作为参数并把它转换为元组。如果参数是元组，那么参数就会被原样返回。如下输入：

```
>>> tuple(['hello','world'])
```

```
('hello', 'world')
```

```
>>> tuple('hello')
```

```
('h', 'e', 'l', 'l', 'o')
```

```
>>> tuple(('hello','world')) #参数是元组
```

```
('hello', 'world')
```

tuple函数传入元组参数后，得到的返回值就是传入参数。

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(3) 元组

1) 访问元素

元组可以使用**下标索引**来访问元组中的值，例如：

```
>>> mix = ('hello', 'world', 2015, 2016)
```

```
>>> print ("mix[1] is: ", mix[1])
```

```
mix[1] is: world
```

```
>>> num = (1, 2, 3, 4, 5, 6, 7 )
```

```
>>> print ("num[1:5] is: ", num[1:5])
```

```
num[1:5] is: (2, 3, 4, 5)
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(3) 元组

2) 连接组合

元组中的元素值是不允许修改的，但我们可以对元组进行连接组合。

例如：

```
>>> field = ('hello', 'world')
```

```
>>> num = (2015, 2016)
```

```
>>> print ("合并结果为：", field+num)
```

合并结果为： ('hello', 'world', 2015, 2016)

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(3) 元组

3) 删除元组

元组中的元素值是不允许删除的，但我们可以使用del语句来删除整个元组。

例如：

```
>>> field = ('hello', 'world')
```

```
>>> del field
```

```
>>> print('删除后的结果:', field)
```

```
Traceback (most recent call last):
```

```
File "<pyshell#84>", line 1, in <module>
```

```
    print('删除后的结果:', field)
```

```
NameError: name 'field' is not defined
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(3) 元组

4) 截取

因为元组也是一个序列，所以我们可以访问元组中的指定位置的元素，也可以截取索引中的一段元素

```
>>> field = ('hello', 'world', 'welcome')
```

```
>>> field [2]
```

```
'welcome'
```

```
>>> field [-2]
```

```
'world'
```

```
>>> field [1:]
```

```
('world', 'welcome')
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(3) 元组

5) 嵌套元组

在元组中可以嵌套元组，在元组中嵌套的元组取出后还是元组。

```
>>> field=('a', 'b', 'c')
```

```
>>> num=(1, 2, 3)
```

```
>>> mix=(field, num)
```

```
>>> mix
```

```
(( 'a', 'b', 'c'), (1, 2, 3))
```

```
>>> mix[0]
```

```
('a', 'b', 'c')
```

```
>>> mix[1]
```

```
(1, 2, 3)
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

Python中对字典进行了构造，让我们可以轻松查到某个特定的**键**（类似拼音或笔画索引），从而通过键找到对应的**值**（类似具体某个字）。在其它语言中称为Hashtable（哈希表）。

字典的创建格式如下：

```
>>> d = {key1 : value1, key2 : value2 }
```

字典由多个键及与其对应的值构成的对组成（把键/值对称为项）。字典的每个键/值(key/value)对用**冒号(:)**分割，每个项之间用**逗号(,)**分割，整个字典包括在花括号({})中。空字典（不包括任何项）由两个大括号组成，如：{ }。

键必须是唯一的，但值则不必。值可以取任何数据类型，但**键必须是不可变的**，如字符串、数字或元组。

```
>>> dict = {'小萌': '1001', '小智': '1002', '小强': '1003'}
```

```
或>>> dict2 = { 'abc': 123, 98.6: 37 }
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

也可以用`dict`函数，通过其他映射（比如其他字典）或者（键/值）这样的序列对建立字典。看如下输入：

```
>>> student=[('name', '小萌'), ('number', '1001')]
```

```
>>> detail=dict(student)
```

```
>>> print('学生详细信息: ', detail)
```

```
学生详细信息:  {'name': '小萌', 'number': '1001'}
```

```
>>> print('学生姓名: ', detail['name'])
```

```
学生姓名:  小萌
```

```
>>> print('学生学号: ', detail['number'])
```

```
学生学号:  1001
```


第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

1) 修改字典

向字典添加新内容的方法是增加新的键/值对，修改或删除已有键/值对。如下示例：

```
>>> student={'小萌':'1001','小智':'1002','小强':'1003'}
```

```
>>> student['小强']='1005' #更新小强的学号
```

```
>>> print('小强的学号是：', student['小强'])
```

小强的学号是：1005

```
>>> student['小张']='1006' #添加一个学生
```

```
>>> print('小张的学号是：%(小张)s' % student)
```

小张的学号是：1006

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

2) 删除字典元素

此处的删除指的是显式删除，显式删除一个字典用del命令，如下示例：

```
>>> student={'小强': '1005', '小萌': '1001', '小智': '1002',  
'小张': '1006'}
```

```
>>> print('删除前:', student)
```

```
删除前: {'小强': '1005', '小萌': '1001', '小智': '1002', '小  
张': '1006'}
```

```
>>> del student['小张'] #删除键 “小张”
```

```
>>> print('删除后:', student)
```

```
删除后: {'小强': '1005', '小萌': '1001', '小智': '1002' }
```

除了删除键，也可以删除整个字典

```
>>> del student
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

3) len函数

len(dict)，该函数用于计算字典元素个数，即键的总数。例如：

```
>>> student={'小萌': '1001', '小智': '1002', '小强':  
'1005', '小张': '1006'}
```

```
>>> print('字典元素个数为：%d个' % len(student))
```

字典元素个数为：4个

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

4) `type`函数

`type(variable)`，该函数返回输入的变量类型，如果输入变量是字典就返回字典类型。例如：

```
>>> student={'小萌': '1001', '小智': '1002', '小强':  
'1005', '小张': '1006'}
```

```
>>> print('字典的类型为: ', type(student))
```

```
字典的类型为:  <class 'dict'>
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

5) `clear`函数

`dict.clear()`，该函数删除字典内所有的项，返回`None`。例如：

```
>>> student={'小萌': '1001', '小智': '1002', '小强':  
'1005', '小张': '1006'}
```

```
>>> student.clear()
```

```
>>> print('字典删除后元素个数为: %d个' % len(student))  
字典删除后元素个数为: 0个
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

6) `copy`函数

`dict.copy()`，返回一个具有相同键/值对的新字典。例如：

```
>>> student={'小萌': '1001', '小智': '1002', '小强':  
'1005', '小张': '1006'}
```

```
>>> st=student.copy()
```

```
>>> print('复制后得到的st为:', st)
```

复制后得到的st为: {'小强': '1005', '小萌': '1001', '小智':
'1002', '小张': '1006'}

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

7) **fromkeys**函数

`dict.fromkeys(seq[, value])`，用于创建一个新字典，以序列`seq`中元素做字典的键，`value`为字典所有键对应的初始值。例如：

```
>>> seq = ('name', 'age', 'sex')
```

```
>>> info = dict.fromkeys(seq)
```

```
>>> print ("新的字典为 :", info)
```

```
新的字典为 : {'name': None, 'sex': None, 'age': None}
```

```
>>> info = dict.fromkeys(seq, 10)
```

```
>>> print ("新的字典为 : %s" % info)
```

```
新的字典为 : {'name': 10, 'sex': 10, 'age': 10}
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

8) **in**操作

key in dict, 用于判断键是否存在于字典中, 如果键在字典dict里返回true, 否则返回false。例如:

```
>>> student={'小萌': '1001', '小智': '1002'}
```

```
>>> print('小萌在student字典中: %s'%('小萌' in student))
```

```
小萌在student字典中: True
```

```
>>> print('小强在student字典中: %s'%('小强' in student))
```

```
小强在student字典中: False
```


第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

9) **items**方法

`dict.items()`，以列表返回可遍历的(键, 值)元组数组。例如：

```
>>> student={'小萌': '1001', '小智': '1002'}
```

```
>>> print('调用items方法的结果: %s'% student.items())
```

```
调用items方法的结果: dict_items([('小萌', '1001'), ('小智',  
'1002')])
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

10) **keys**方法

`dict.keys()`，以列表返回一个字典所有的键。例如：

```
>>> student={'小萌': '1001', '小智': '1002'}
```

```
>>> print('字典student所有的键为：%s'% student.keys())
```

```
字典student所有的键为: dict_keys(['小萌', '小智'])
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

11) **setdefault**方法

`dict.setdefault(key, default=None)`, `setdefault()`方法和`get()`方法类似, 就是获得与给定键相关联的值, 如果键不存在于字典中, 将会添加键并将值设为默认值。例如:

```
>>> student={'小萌': '1001', '小智': '1002'}
```

```
>>> print('小强的键值为: %s'% student.setdefault('小强'))
```

小强的键值为: None

```
>>> print('小智的键值为: %s'% student.setdefault('小智'))
```

小智的键值为: 1002

```
>>> print('student字典新值为: %s'% student)
```

student字典新值为: {'小强': None, '小萌': '1001', '小智': '1002'}

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

12) **update**方法

`dict.update(dict2)`，把字典`dict2`的键/值对更新到`dict`里。例如：

```
>>> student={'小萌': '1001', '小智': '1002'}
```

```
>>> student2={'小李': '1003'}
```

```
>>> print('原student字典为: %s'% student)
```

```
原student字典为: {'小萌': '1001', '小智': '1002'}
```

```
>>> student.update(student2)
```

```
>>> print('新student字典为: %s'% student)
```

```
新student字典为: {'小萌': '1001', '小智': '1002', '小李':  
'1003'}
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

13) **values**方法

`dict.values()`，以列表形式返回字典中的所有值。例如：

```
>>> student={'小萌': '1001', '小智': '1002', '小李': '1001'}  
>>> print('student字典所有值为: %s'% list(student.values()))  
student字典所有值为: ['1001', '1001', '1002']
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

字典键的特性

两个重要的点需要记住：

1) 不允许同一个键出现两次。看如下示例：

```
>>> student={'小萌': '1001', '小智': '1002', '小萌': '1005'}
>>> print('学生信息:', student)
```

学生信息: {'小萌': '1005', '小智': '1002'}

2) 键必须不可变，所以可以用数字，字符串或元组充当，而用列表就不行，看如下示例：

```
>>> field={['name']: '小萌', 'number': '1001'}
```

Traceback (most recent call last):

File "<pyshell#80>", line 1, in <module>

```
field={['name']: '小萌', 'number': '1001'}
```

TypeError: unhashable type: 'list'

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(4) 字典

字典和列表的区别：

dict有以下几个特点

- (1) 查找和插入的速度极快，不会随着key的增加而变慢；
- (2) 需要占用大量的内存，内存浪费多。

List的特点是

- (1) 查找和插入的时间随着元素的增加而增加；
- (2) 占用空间小，浪费内存很少。

dict使用的是用空间来换取时间。

dict可以用在需要高速查找的很多地方，需要牢记的第一条就是dict的key必须是不可变对象。

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(5) 集合

集合（**set**）是一种无序集，它是一组键的集合，不存储值。

在集合中，重复的键是不被允许的。**集合可以用于去除重复值。**

集合也可以进行数学集合运算，如**并、交、差**以及**对称差**等。

```
>>>x = set('runoob')
>>> y = set('google')
>>> x, y
(set(['b', 'r', 'u', 'o', 'n']), set(['e', 'o', 'g', 'l'])) # 重复的被删除
>>> x & y          # 交集
set(['o'])
>>> x | y          # 并集
set(['b', 'e', 'g', 'l', 'o', 'n', 'r', 'u'])
>>> x - y          # 差集
set(['r', 'b', 'u', 'n'])
```


第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(5) 集合

集合的创建有两种方式：使用 `set()` 函数或者使用大括号 `{}`。

需要注意的是，创建空集合，必须使用 `set()`，而不是 `{}`，因为 `{}` 表示创建一个空的字典。

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(5) 集合

集合支持数学集合运算，如并、交、差以及对称差等。

```
shot_type_set1 = {'Jump Shot', 'Running Jump Shot', 'Driving Dunk Shot'}
shot_type_set2 = {'Layup Shot', 'Jump Shot', 'Driving Layup Shot'}
print '\t',shot_type_set1
print '\t',shot_type_set2
print '并集 (或) ', '\n\t',shot_type_set1|shot_type_set2
print '交集 (与) ', '\n\t',shot_type_set1&shot_type_set2
print '差集', '\n\t',shot_type_set1-shot_type_set2
print '对称差 (异或) ', '\n\t',shot_type_set1^shot_type_set2

set(['Driving Dunk Shot', 'Running Jump Shot', 'Jump Shot'])
set(['Jump Shot', 'Driving Layup Shot', 'Layup Shot'])
并集 (或)
set(['Running Jump Shot', 'Jump Shot', 'Driving Dunk Shot', 'Driving Layup Shot', 'Layup Shot'])
交集 (与)
set(['Jump Shot'])
差集
set(['Driving Dunk Shot', 'Running Jump Shot'])
对称差 (异或)
set(['Driving Dunk Shot', 'Running Jump Shot', 'Driving Layup Shot', 'Layup Shot'])
```

第2章 Python编程基础

2.3 数据类型——列表、元组、字典和集合

(5) 集合

Python的集合运算

函数	其他表示法	说明
a.add(x)	N/A	把元素x添加到集合a中
a.remove(x)	N/A	把元素x从集合a中删除
a.union(b)	a b	a和b中全部的唯一元素
a.intersection(b)	a & b	a和b都有的元素
a.difference(b)	a - b	a中不属于b的元素
a.symmetric_difference(b)	a ^ b	a或b中不同时属于a和b的元素
a.issubset(b)	N/A	如果a的全部元素都包含于b，则为True
a.issuperset(b)	N/A	如果b的全部元素都包含于a，则为True
a.isdisjoint(b)	N/A	如果a和b没有公共元素，则为True

第2章 Python编程基础

2.3 数据类型——转换函数

函数	描述
<u>int(x [,base])</u>	将x转换为一个整数
<u>long(x [,base])</u>	将x转换为一个长整数
<u>float(x)</u>	将x转换到一个浮点数
<u>complex(real [,imag])</u>	创建一个复数
<u>str(x)</u>	将对象 x 转换为字符串
<u>repr(x)</u>	将对象 x 转换为表达式字符串
<u>eval(str)</u>	用来计算在字符串中的有效Python表达式,并返回一个对象
<u>tuple(s)</u>	将序列 s 转换为一个元组
<u>list(s)</u>	将序列 s 转换为一个列表
<u>set(s)</u>	转换为可变集合

第2章 Python编程基础

2.3 数据类型——转换函数

函数	描述
<u>dict(d)</u>	创建一个字典。 d 必须是一个序列 (key,value)元组。
<u>frozenset(s)</u>	转换为不可变集合
<u>chr(x)</u>	将一个整数转换为一个字符
<u>unichr(x)</u>	将一个整数转换为 Unicode 字符
<u>ord(x)</u>	将一个字符转换为它的整数值
<u>hex(x)</u>	将一个整数转换为一个十六进制字符串
<u>oct(x)</u>	将一个整数转换为一个八进制字符串

第2章 Python编程基础

2.3 数据类型——空值判断

Python中对变量是否为None的判断

三种主要的写法有：

第一种： `if X is None;`

第二种： `if not X;`

当X为None, False, 空字符串 "", 0, 空列表 [], 空字典 {}, 空元组 () 这些时, not X为真, 即无法分辨出他们之间的不同。

第三种： `if not X is None;`

在Python中, None、空列表 []、空字典 {}、空元组 ()、0 等一系列代表空和无的对象会被转换成False。除此之外的其它对象都会被转化成True。

在命令 `if not 1` 中, 1 便会转换为bool类型的True。not是逻辑运算符非, not 1 则恒为False。因此if语句 `if not 1` 之下的语句, 永远不会执行。

第2章 Python编程基础

2.4 输入、输出和注释语句

```
>>> userName = input("输入登录名: ")
```

输入登录名: 周星星

```
>>> print("你的登录名为", userName)
```

你的登录名为 周星星 # 自动加空格

input(): 从用户那里得到数据输入;

```
>>> myNumber = input("输入一个数字:")
```

输入一个数字:1024

```
>>> print "你输入数字的2倍是: %d" % (int(myNumber) * 2)
```

你输入数字的2倍是: 2048

int() 字符串转换为整型

第2章 Python编程基础

2.5 赋值语句

等号（=）用来给变量赋值。

等号（=）运算符左边是一个变量名，等号（=）运算符右边是存储在变量中的值。

每个变量在**内存**中创建，都包括变量的标识，名称和数据这些信息。

每个变量在**使用前都必须赋值**，**变量赋值以后该变量才会被创建**。

Python允许同时为**多个变量赋值**。例如：

```
>>> a = b = c = 1
```

```
>>> a, b, c = 1, 2, " John "
```


第2章 Python编程基础

2.5 赋值语句

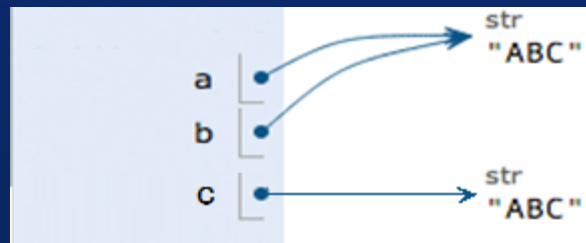
Python变量名规则与其他编程语言一样，并且**大小写敏感**

```
>>> pptname = "Introction to Python"
>>> pptName = "Python入门"
>>> height = 1.71
>>> age = 26
>>> n = height
>>> n *= 100 # 等价于 n = n * 100
```

第2章 Python编程基础

2.5 赋值语句

- (1) 执行`a = 'ABC'`，解释器创建了字符串'ABC'和变量`a`，并把`a`指向'ABC'。
- (2) 执行`b = a`，解释器创建了变量`b`，并把`b`指向`a`指向的字符串'ABC'。
- (3) 执行`c = 'ABC'`，解释器创建了字符串'ABC'和变量`c`，并把`a`指向'ABC'，但它和`a`、`b`没有直接关系。
- (4) 执行`a = 'XYZ'`，解释器创建了字符串'XYZ'，并把`a`的指向改为'XYZ'，但`b`并没有更改。



第2章 Python编程基础

2.6 条件语句

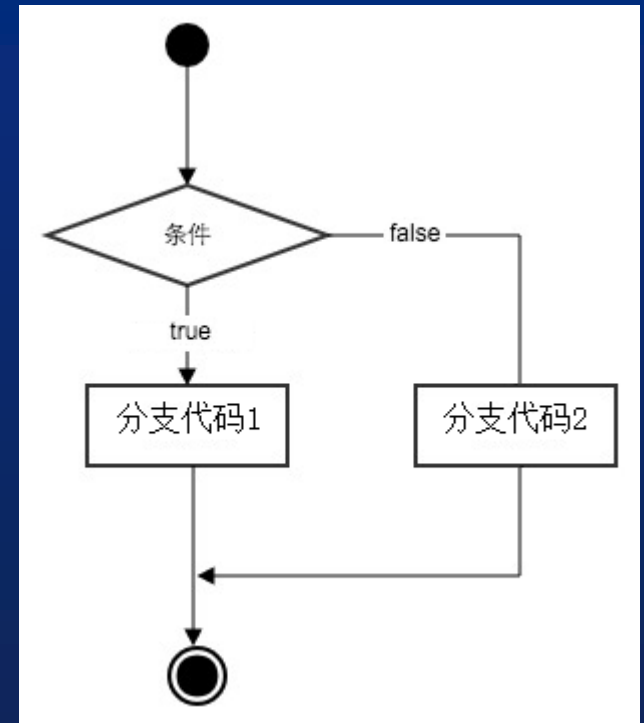
Python条件语句是通过一条或多条语句的执行结果（True或者False）来决定执行的代码块。

Python程序语言指定任何非0和非空（null）值为true，0 或者 null为false。

Python 编程中 if 语句用于控制程序的执行，基本形式为：

```
if 判断条件：  
    执行语句……
```

```
else:  
    执行语句……
```



第2章 Python编程基础

2.7 循环语句

(1) while循环

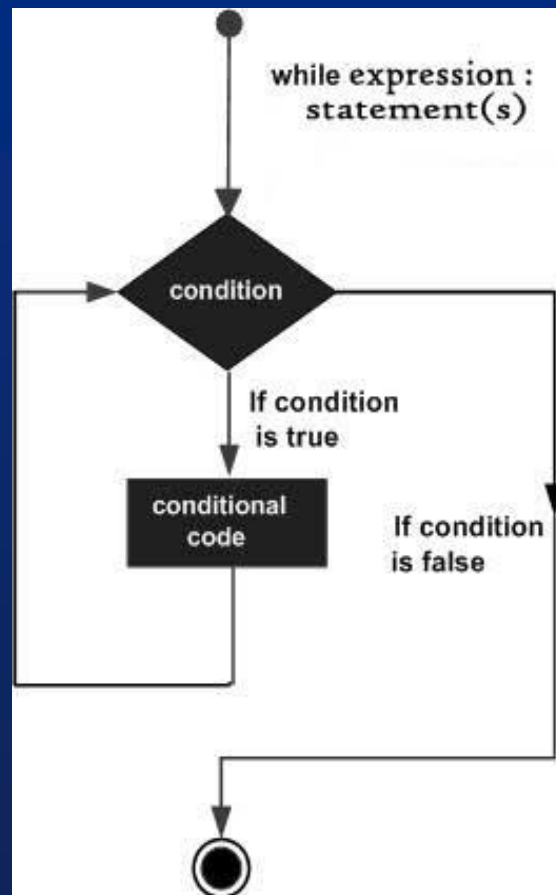
Python编程中while用于循环执行程序，在某条件下，循环执行某段程序，处理需要重复处理的相同任务。其语法形式为：

while 判断条件：

 执行语句……

执行语句可以是单个语句或语句块。判断条件可以是任何表达式，任何非零、或非空（null）的值均为真（true）。当判断条件为假（false）时，循环结束。

while循环的执行流程图如右图所



第2章 Python编程基础

2.7 循环语句

(2) for循环

在Python中，for关键字叫做for循环，for循环可以遍历任何序列的项目，如一个列表或者一个字符串。

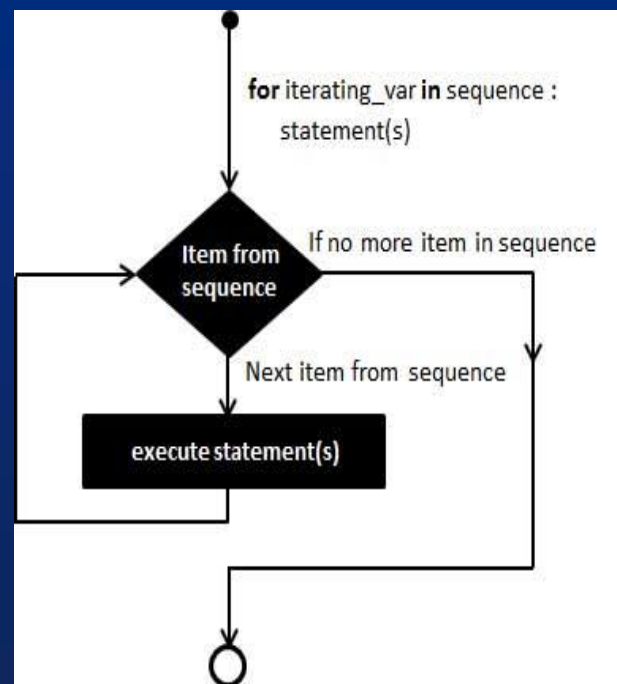
for循环的语法格式如下：

```
for iterating_var in sequence:  
    statements(s)
```

sequence是任意序列，

iterating_var是需要遍历的元素。

statements是待执行的语句块。

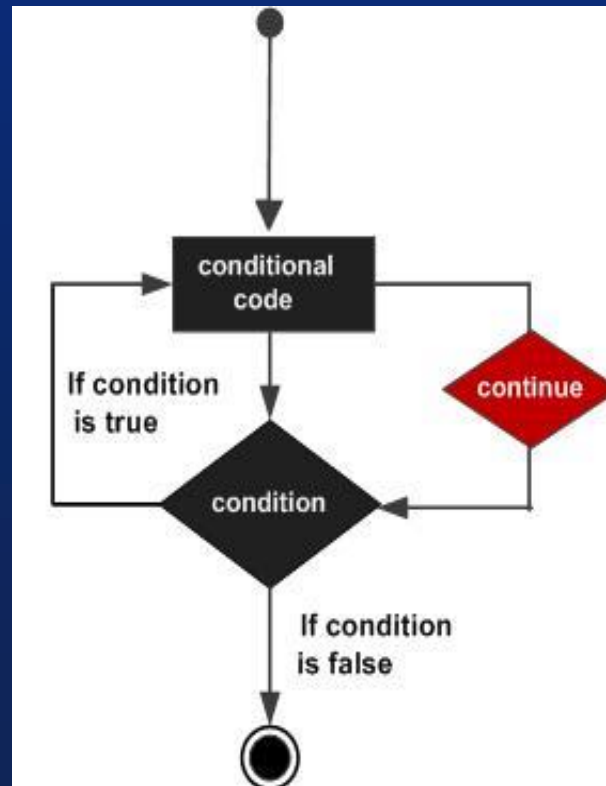
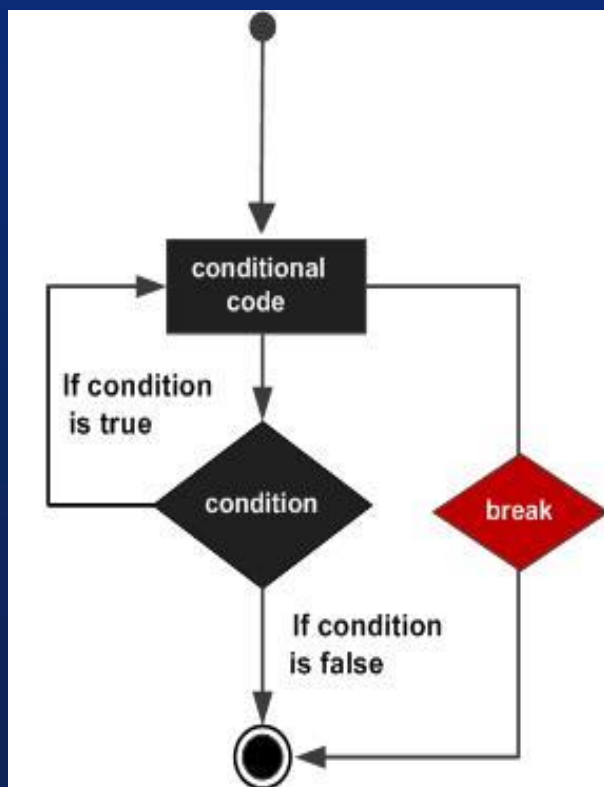


第2章 Python编程基础

2.7 循环语句

(3) 跳出循环

Python中提供了break、continue等语句可用于跳出循环。



第2章 Python编程基础

2.7 循环语句

(4) 循环中的else子句

1) while循环使用else语句

在while条件语句为false时执行else的语句块，如下所示：

```
num = 0
while num < 3:
    print (num, " 小于 3")
    num = num + 1
else:
    print (num, " 大于或等于 3")
print("结束循环!")
```

第2章 Python编程基础

2.7 循环语句

(4) 循环中的else子句

1) for循环使用else语句

在for条件语句为false或结束后**没有被break中断时**执行else的语句块，如下所示：

```
names = ['xiaomeng', 'xiaozhi']
for name in names:
    if name == "xiao":
        print("名称: ", name)
        break
    print("循环名称列表 " + name)
else:
    print("没有循环数据!")
print("结束循环!")
```

```
循环名称列表 xiaomeng
循环名称列表 xiaozhi
没有循环数据!
结束循环!
```


第2章 Python编程基础

2.8 pass语句

Python中的pass是空语句，其作用是为了保持程序结构的完整性。

pass语句语法格式如下：

```
pass
```

pass不做任何事情，一般用做占位语句。看如下代码：

```
>>> pass
```

```
>>>
```

输出结果什么都没有做。

The End

