第 3 章

Python 数据类型

本章首先介绍了变量的命名和引用，以及各类运算符，如算术运算符、关系运算符、逻辑运算符、身份运算符等；然后重点介绍了序列、字典和集合等数据类型的相关内容和常用操作，其中，序列类型包括字符串、列表、元组等是具有顺序特征的数据类型；最后，介绍了数据类型转换的相关知识。

Python 数据类型

3.1 变量

在 Python 中，每个变量在使用之前都必须赋值，变量只有在赋值之后才会被创建。

3.1.1 变量命名

变量的命名必须遵循以下规则。

(1) 变量名由字母、数字和下画线组成。

(2) 变量名的第一个字符必须是字母或者下画线，不能以数字开头。

下面的变量命名不符合变量命名规则，导致语法错误，如图 3-1 所示。

```
>>> 3xy
  File "<stdin>", line 1
    3xy
     ^
SyntaxError: invalid syntax
>>> ab%c
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ab' is not defined
>>> Wang ping
  File "<stdin>", line 1
    Wang ping
         ^
SyntaxError: invalid syntax
```

图 3-1 错误变量命名示例

(3) 尽量不要使用容易混淆的单个字符作为标识符。例如，数字 0 和字母 o，数字 1 和字母 l 等。

(4) 变量名不能与关键字同名。在 Anaconda Prompt 下输入命令“import keyword”查看 Python 的关键字，如图 3-2 所示。

(5) 变量名区分大小写。例如，myname 和 myName 不是同一个变量。

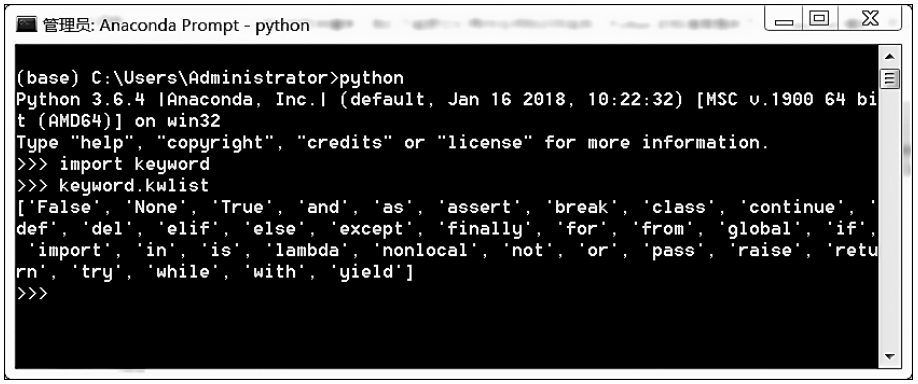

图 3-2 Python 的关键字

(6) 以双下画线开头的标识符是有特殊意义的,是 Python 采用特殊方法的专用标识,如__init__()代表类的构造函数。

(7) Python 中,单独的下画线用于表示上一次运算的结果。

例如:

```
>>>20
20
>>>_*10
200
```

(8) 变量名和函数名一般用英文小写字母,以增加程序的可读性。

(9) 变量命名应见名知义,通过变量名知道变量的含义,一般选用英文单词或拼音缩写的形式,如求“和”用“sum”,不要使用简单符号,如 x、y、z 等。

3.1.2 变量引用

Python 中的变量通过赋值得到值。

【例 3-1】 变量引用。

```
>>>number =5
>>>number
5
>>>number =7
>>>print(number)
7
```

3.2 运算符

变量之间的运算可以通过运算符实现,运算符包括算术运算符、关系运算符、赋值运算符、逻辑运算符、位运算符、成员运算符和身份运算符等。

3.2.1 算术运算符

算术运算符如表 3-1 所示。

表 3-1　算术运算符

运算符	含义	运算符	含义
＋	加法	//	取整除
－	减法	* *	幂运算
*	乘法	%	取模
/	除法		

运算符的使用和运算数的数据类型关系很大，加法运行结果如图 3-3 所示。

```
>>> print(10+3)
13
>>> print('a'+'b')
ab
>>> print(a+b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

图 3-3　加法运行结果

【例 3-2】 算术运算符。

下面给出除法(/)、整除(//)、求余数(%)的运行结果，如图 3-4 所示。

```
>>> print(10/3)
3.3333333333333335
>>> print(10//3)
3
>>> print(10%3)
1
```

图 3-4　除法(/)、整除(//)、求余数(%)的运行结果

3.2.2　关系运算符

关系运算符又称为比较运算符，是双目运算符，用于两个操作数的大小比较，结果是布尔值，即 True(真)或 False(假)。操作数可以是数值型或字符型。表 3-2 列出了 Python 中的关系运算符。

表 3-2　关系运算符

运算符	描　　述	运算符	描　　述
==	等于	<	小于
>	大于	<=	小于或等于
>=	大于或等于	! =	不等于

关系运算符在进行比较时，需注意以下规则。

(1) 两个操作数是数字，按大小进行比较。需要注意的是，Python 中“==”是等于号，“! =”是不等于号，如图 3-5 所示。

(2) 两个操作数是字符型，按字符的 ASCII 码值从左到右逐一比较，首先比较两个字符

```
>>> print(3<5)
True
>>> print(3=5)
  File "<stdin>", line 1
SyntaxError: keyword can't be an expression
>>> print(3==5)
False
>>> print(3>5)
False
>>> print(3!=5)
True
>>> print(3<>5)
  File "<stdin>", line 1
    print(3<>5)
          ^
SyntaxError: invalid syntax
>>>
```

图 3-5　操作数为数字的运行结果

串的第一个字符，ASCII 码值大的字符串为大，如果第一个字符相同，比较第二个字符，以此类推，直到出现不同的字符为止，如图 3-6 所示。

```
>>> print("abc"=="abcd")
False
>>> print("abcd"=="abcd")
True
>>> print("abc">"abd")
False
>>> print("abc"<"abd")
True
>>> print("23"<"3")
True
>>> print("abc"!="abc")
False
>>> print("abc"!="ABC")
True
```

图 3-6　操作数为字符串的运行结果

3.2.3　赋值运算符

赋值运算符如表 3-3 所示。

表 3-3　赋值运算符

运算符	描　　述	运算符	描　　述
=	简单赋值运算符	/=	除法赋值运算符
+=	加法赋值运算符	%=	取模赋值运算符
-=	减法赋值运算符	**=	幂赋值运算符
*=	乘法赋值运算符	//=	取整除赋值运算符

【例 3-3】 赋值运算符。

赋值运算符举例如图 3-7 所示。

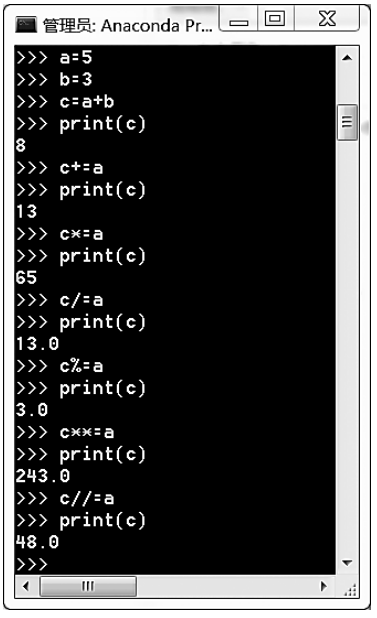

图 3-7　赋值运算符举例

3.2.4　逻辑运算符

逻辑运算符如表 3-4 所示。not 是单目运算符，其余都是双目运算符，逻辑运算的结果是布尔值 True 或 False。

表 3-4　逻辑运算符

运算符	含义	描　　述
not	取反	当操作数为假时，结果为真；当操作数为真时，结果为假
and	与	当两个操作数均为真时，结果为真；否则为假
or	或	当两个操作数至少有一个为真时，结果为真；否则为假

【例 3-4】　逻辑运算符。

逻辑运算符举例如图 3-8 所示。

注意：False 不能写成 F、false 等。

3.2.5　位运算符

位运算是把十进制数字转换为二进制数字的运算。Python 中的位运算符有左移(<<)、右移(>>)、按位与(&)、按位或(|)、按位翻转(~)等，如表 3-5 所示。

表 3-5　位运算符

运算符	名　称	描　　述
<<	左移	把一个数的二进制数字向左移一定数目
>>	右移	把一个数的二进制数字向右移一定数目

续表

运算符	名　称	描　　述
&	按位与	数的按位与
|	按位或	数的按位或
^	按位异或	数的按位异或
~	按位翻转	x 的按位翻转是－(x＋1)

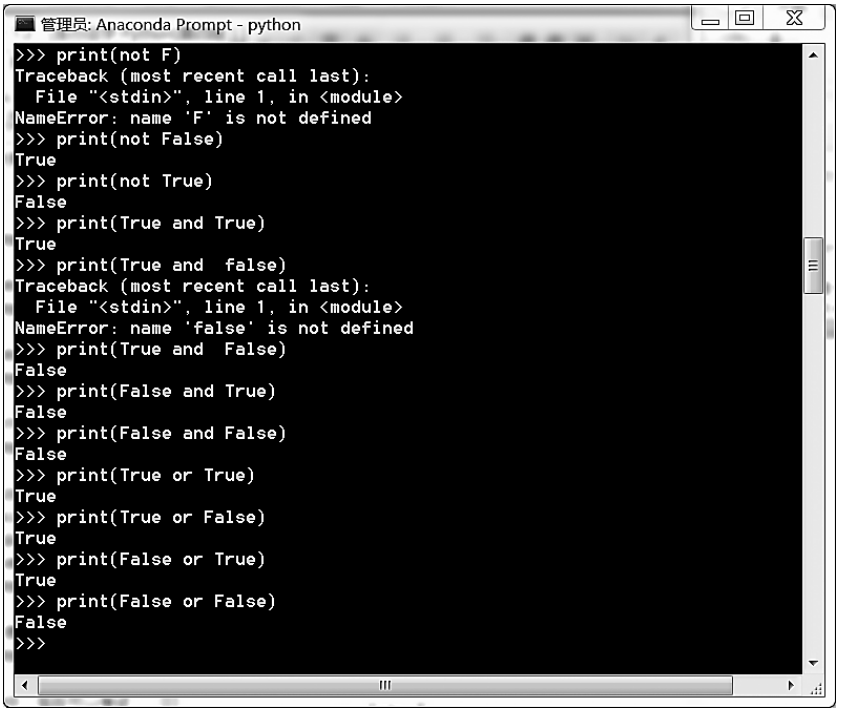

图 3-8　逻辑运算符举例

【例 3-5】 位运算符。

位运算符举例如图 3-9 所示。

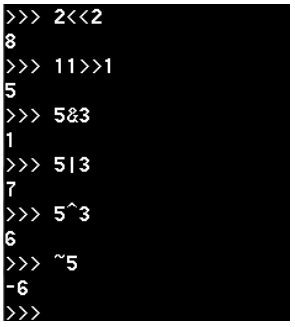

图 3-9　位运算符举例

3.2.6 成员运算符

成员运算符主要用于字符串、列表或元组等序列数据类型，如表 3-6 所示。

表 3-6 成员运算符

运算符	描 述
in	在指定的序列中找到值返回 True，否则返回 False
not in	在指定的序列中没有找到值返回 True，否则返回 False

【例 3-6】 成员运算符。

```
>>>'a' not in 'bcd'
   True
>>>3 in[1,2,3,4]
   True
```

3.2.7 身份运算符

身份运算符又称为同一运算符，用于比较两个对象的存储单元，如表 3-7 所示。

表 3-7 身份运算符

运 算 符	描 述
is	判断两个标识符是不是引用自一个对象
is not	判断两个标识符是不是引用自不同对象

【例 3-7】 身份运算符。

```
>>>x=y=3.5
>>>z=3.5
>>>x is y
True
>>>x is z
False
>>>x is not z
True
```

3.3 表达式

3.3.1 概念

表达式通常由运算符号(操作符)和参与运算的数(操作数)两部分组成。例如，2＋3 就是一个表达式。＋就是运算符，2 和 3 就是操作数。

数学表达式转换为 Python 表达式，如表 3-8 所示。

表 3-8 数学表达式转换为 Python 表达式

数学表达式	Python 表达式
$\frac{abcd}{efg}$	$a*b*c*d/e/f/g$ 或 $a*b*c*d/(e*f*g)$
$\sin45+\frac{e^{10}+\ln10}{\sqrt{x}}$	math.sin(45 * 3.14/180)＋(math.exp(10)＋math.log(10))/math.sqrt(x)
$[(3x+y)-z]^{1/2}/(xy)^4$	math.sqrt((3 * x＋y)－z)/(x * y)^4

数学表达式转换为 Python 表达式应注意如下区别。

(1) 乘号不能省略。例如,x 乘以 y 写成 Python 表达式为 x * y。

(2) 括号必须成对出现,均使用圆括号,出现多个圆括号时,从内向外逐层配对。

(3) 运算符不能相邻。例如,a＋－b 是错误的。

(4) 添加必要的函数。例如,数学表达式$\sqrt{25}$转换为 Python 表达式为 sqrt(25)。

3.3.2 操作

表达式计算根据运算符的优先次序逐一进行计算,Python 运算符的优先级如表 3-9 所示。

表 3-9 Python 运算符的优先级别

优先级	运 算 符	描 述
高↑低	* *	指数
	～、＋、－	按位翻转、正号、负号
	* 、/、%、//	乘法、除法、取模和取整除
	＋、－	加法、减法
	＞＞、＜＜	右移、左移
	&.	按位与
	^、|	按位异或、按位或
	＜＝、＜、＞、＞＝	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

3.4 数据类型

Python 3 具有 6 个标准的数据类型：Number(数字),String(字符串),List(列表),Tuple(元组),Dictionary(字典),Set(集合)。其中,不可变的数据类型有 Number、String 和

Tuple。可变的数据类型有 List、Dictionary 和 Set。

3.5　数字

3.5.1　概念

Python 中数值有 4 种类型：整型、布尔型、浮点数和复数。

(1) 整型(int)——表示整数。例如，1，1024，－982。

(2) 布尔型(bool)——表示布尔逻辑值。例如，True，False。

(3) 浮点数(float)——表示小数。例如，1.23，3.14，－9.01 等。之所以称为浮点数，是因为按照科学记数法表示，浮点数的小数点位置可变。例如，53.3E4 就是科学记数法，E 表示 10 的幂，53.3E4 表示 53.3×10^4。53.3E4 和 5.33E5 表示同一数字，但是它们的小数点位置不同。

(4) 复数(complex)——表示复数。例如，1＋2j，1.1＋3.2j。

3.5.2　操作

【例 3-8】 数值。

```
>>>a, b, c, d =20, 5.5, True, 4+3j
>>>print(type(a), type(b), type(c), type(d))
<class 'int'><class 'float'><class 'bool'><class 'complex'>
```

数学函数如表 3-10 所示。

表 3-10　数学函数

函　数	含　义	举　例
abs(x)	数字的绝对值	math.abs(－10) 返回 10
ceil(x)	数字的上入整数	math.ceil(4.1) 返回 5
exp(x)	e 的 x 次幂(e^x)	math.exp(1) 返回 3.718 281 828 459 045
fabs(x)	数字的绝对值	math.fabs(－10) 返回 10.0
floor(x)	数字的下舍整数	math.floor(4.9)返回 4
log(x)	x 的对数	math.log(100,10)返回 2.0
log10(x)	以 10 为基数的 x 的对数	math.log10(100)返回 2.0
max(x1, x2,…)	给定参数的最大值，参数可以为序列	math.max(2,3)返回 3
min(x1, x2,…)	给定参数的最小值，参数可以为序列	math.min(2,3)返回 2
pow(x, y)	x * * y 运算后的值	math.pow(2,3) 返回 8
round(x [,n])	浮点数 x 的四舍五入值，n 代表舍入到小数点后的位数	math.round(3.4)返回 3
sqrt(x)	数字 x 的平方根	math.sqrt(4) 返回 2


字符串

3.6 字符串

3.6.1 概念

字符串在 Python 中是以单引号、双引号或三引号括起来的符号来表示，例如，'Hello'、"Python is groovy"、"""What is footnote 5?"""等。

注意，"或""本身只是一种表示方式，不是字符串的一部分。因此，字符串'abc'中只有 a，b，c 三个字符。另外，用单引号或双引号括起来没有任何区别，只是一个字符串用什么引号开头，就必须用什么引号结尾。

单引号与双引号只能创建单行字符串，为了创建多行字符串或者为了使得字符串的数据中出现双引号，则出现了三引号，如图 3-10 所示。

```
>>> 'Hello'
'Hello'
>>> "let's go"
"let's go"
>>> '''Life is short
We need Python'''
'Life is short\nWe need Python'
```

图 3-10 单引号与双引号举例

3.6.2 操作

字符串(String)与列表和元组都是序列，其方法如表 3-11 所示。

表 3-11 字符串方法

函 数	描 述
s.index(sub,[start,end])	定位子串 sub 在 s 里第一次出现的位置
s.find(sub,[start,end]})	与 index 函数一样，但如果找不到会返回－1
s.replace(old, new [,count])	替换 s 里所有 old 子串为 new 子串，count 指定多少个字符可被替换
s.count(sub[,start,end])	统计 s 里有多少个 sub 子串
s.split( )	默认分隔符是空格
s.join()	join()方法是 split()方法的逆方法，用来把字符串连接起来
s.lower( )	返回将大写字母变成小写字母的字符串
s.upper( )	返回将小写字母变成大写字母的字符串
sep.join(sequence)	把 sequence 的元素用连接符 sep 连接起来

下面介绍字符串的操作。

1. index()方法举例

```
>>>s="Python"
>>>s.index('P')
```

```
0
>>>s.index('h',1,4)
3
>>>s.index('y',3,4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>>s.index('h',3,4)
3
```

2. find()方法举例

```
>>>s="Python"
>>>s.find('s')
-1
>>>s.find('t',1)
2
```

3. replace()方法举例

```
>>>s="Python"
>>>s.replace('h','i')
'Pytion'
```

4. count()方法举例

```
>>>s="Python"
>>>s.count('n')
1
```

5. split()方法举例

```
>>>s="Python"
>>>s.split()
['Python']
>>>s="hello Python i like it"
>>>s.split()
['hello', 'Python', 'i', 'like', 'it']
>>>s ='name:zhou,age:20|name:python,age:30|name:wang,age:55'
>>>print(s.split('|') )
['name:zhou,age:20','name:python,age:30','name:wang,age:55']
>>>x,y=s.split('|',1)
>>>print(x)
name:haha,age:20
>>>print(y)
name:python,age:30|name:fef,age:55
```

6. join()方法举例

```
>>>li=['apple','peach','banana','pear']
>>>sep=','
>>>s=sep.join(li)                    #连接列表元素
```

```
>>>s
'apple,peach,banana,pear'
>>>s5=("Hello","World")
>>>sep=""
>>>sep.join(s5)                    #连接元组元素
'HelloWorld'
```


列表

3.7 列表

3.7.1 概念

列表(List)是 Python 中使用最频繁的数据类型。列表中的每一个数据称为元素,元素用逗号分隔并放在一对中括号“[”和“]”中,列表可以认为是下标从零开始的数组。列表可以包含混合类型的数据,即在一个列表中的数据类型可以各不相同。

列表举例:

```
[10, 20, 30, 40]                    #所有元素都是整型数据的列表
['frog', 'cat', 'dog']              #所有元素都是字符串的列表
['apple', 3.0, 5, [10, 20], True]   #列表中包含字符串、浮点类型、整型、列表类型、布尔类型
```

Python 创建列表时,解释器在内存中生成一个类似数组的数据结构,数据项自下而上存储,如表 3-12 所示。

表 3-12 列表存储方式

4	True
3	[10,20]
2	5
1	3.0
0	apple

3.7.2 操作

下面介绍列表操作。

1. 创建列表

使用“=”将一个列表赋值给变量。

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

2. 读取元素

(1) 读取某个元素:用列表名加元素序号。

序列中的每个元素被分配一个序号,即元素的位置,也称为索引。从左至右依次是 0,…,n,从右向左计数来存取元素称为负数索引,依次是 $-1,-2,\cdots,-n$。li[$-n$]==li[len(list)$-n$]。

【例 3-9】 列表索引。

```
>>>l1=[1,1.3,"a"]
>>>l1[0]
1
>>>l1[-1]
'a'
```

注意：Python 从 0 开始计数。

(2) 读取若干元素。

序列切片是指使用序列序号截取其中的任何部分从而得到新的序列。切片操作符是在[]内提供一对可选数字，用“:”分隔。冒号前的数字表示切片的开始位置，冒号后的数字表示切片截止(但不包含)位置。

注意：数字可选，冒号必需，开始位置包含在切片中，不包括结束位置。

【例 3-10】 列表切片。

```
>>>l1=[1,1.3,"a"]
>>>l1[1:2]            #取出位置从 1 开始到位置为 2 的字符,但不包含偏移为 2 的元素
[1.3]
>>>l1[:2]             #不指定第一个数,切片从第一个元素直到但不包含偏移为 2 的元素
[1, 1.3]
>>>l1[1:]             #不指定第二个数,从偏移为 1 直到末尾之间的元素
[1.3, 'a']
>>>l1[:]              #数字都不指定,则返回整个列表的一个拷贝
```

3. 修改元素

只需直接给元素赋值。

```
>>>a_list =['a', 'b', 'c']
>>>a_list[0] =123
>>>print a_list
[123, 'b', 'c']
```

4. 添加元素

列表添加元素有“+”、append( )、extend( )和 insert( )方法。

方法 1：使用“+”将一个新列表附加在原列表的尾部。

```
>>>a_list =[1]
>>>a_list =a_list +['a', 3.0]
>>>a_list
[1, 'a', 3.0]
```

方法 2：使用 append( )方法向列表尾部添加一个新元素。

```
>>>a_list =[1, 'a', 3.0]
>>>a_list.append(True)
>>>a_list
[1, 'a', 3.0, True]
```

方法 3：使用 extend( )方法将一个列表添加在原列表的尾部。

```
>>>a_list=[1, 'a', 3.0, True]
>>>a_list.extend(['x', 4])
>>>a_list
[1, 'a', 3.0, True, 'x', 4]
```

方法 4：使用 insert( )方法将一个元素插入列表的任意位置。

```
>>>a_list=[1, 'a', 3.0, True, 'x', 4]
>>>a_list.insert(0, 'x')
>>>a_list
['x', 1, 'a', 3.0, True, 'x', 4]
```

【例 3-11】 比较“+”和 append() 两种方法。

```
import time
result =[]
start =time.time()
for i in range(10000):
    result =result +[i]
print("+操作执行",len(result), '次,用时', time.time()-start)
result =[]
start =time.time()
for i in range(10000):
    result.append(i)
print("append 操作执行",len(result), '次,用时', time.time()-start)
```

程序运行结果如下。

```
+操作执行 10000 次,用时 0.2020115852355957
append 操作执行 10000 次,用时 0.0009999275207519531
```

【例 3-12】 比较 insert() 和 append() 两种方法。

```
import time
def Insert():
    a =[]
    for i in range(10000):
        a.insert(0, i)
def Append():
    a =[]
    for i in range(10000):
        a.append(i)
start =time.time()
for i in range(10):
    Insert()
print('Insert:', time.time()-start)
start =time.time()
```

```
for i in range(10):
    Append()
print('Append:', time.time()-start)
```

程序运行结果如下。

```
Insert: 0.578000068665
Append: 0.0309998989105
```

5. 删除元素

列表删除元素有 del、remove()和 pop()方法。

方法 1：使用 del 语句删除某个特定位置的元素。

```
>>>a_list=['x', 1, 'a', 3.0, True, 'x', 4]
>>>del a_list[1]
>>>a_list
['x', 'a', 3.0, True, 'x', 4]
```

方法 2：使用 remove()方法删除某个特定值的元素。

```
>>>a_list =['x', 'a', 3.0, True, 'x', 4]
>>>a_list.remove('x')
>>>a_list
['a', 3.0, True, 'x', 4]
>>>a_list.remove('x')
>>>a_list
['a', 3.0, True, 4]
>>>a_list.remove('x')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

【例 3-13】 比较两组代码。

```
>>>x =[1,2,1,2,1,2,1,2,1]
>>>for i in x:
        if i ==1:
            x.remove(i)
>>>x
[2, 2, 2, 2]
```

```
>>>x =[1,2,1,2,1,1,1]
>>>for i in x:
            if i ==1:
                x.remove(i)
>>>x
[2, 2, 1]
```

同样的代码，仅仅是处理的列表数据不同，结果不同。两组数据的区别在于有没有连续的“1”。由于列表的自动内存管理功能，删除列表元素，Python 会自动对列表内存进行收缩并移动列表元素，以保证所有元素之间没有空值。增加列表元素时也会自动扩展内存并对元素进行移动，以保证元素之间没有空值。每当插入或删除一个元素之后，该元素位置后面所有元素的索引都会改变。

为此,修改正确的代码如下。

```
>>>x =[1,2,1,2,1,1,1]
>>>for i in x[::]:          #切片
        if i ==1:
            x.remove(i)
```

```
>>>x =[1,2,1,2,1,1,1]
>>>for i in range(len(x)-1,-1,-1):
            if x[i ]==1:
                del x[i]
```

方法 3：使用 pop()方法弹出指定位置的元素,省略参数时弹出最后一个元素。

```
>>>a_list=['a', 3.0, True, 4]
>>>a_list.pop( )                                    #省略参数时弹出最后一个元素
4
>>>a_list
['a', 3.0, True]
>>>a_list.pop(1)
3.0
>>>a_list
['a', True]
>>>a_list.pop(1)
True
>>>a_list
['a']
>>>a_list.pop( )
'a'
>>>a_list
[ ]
>>>a_list.pop( )
Traceback (most recent call last):
  File "<stdin >", line 1, in <module>
IndexError: pop from empty list
```

6. 得到列表中指定元素的下标

```
>>>a=[72, 56, 76, 84, 76, 80, 88]
>>>print(a.index(56))                               #输出元素 56 的下标
1
>>>b=list(enumerate(a))                             #enumerate()将 list 的元素元组化
>>>print(b)
[(0, 72), (1, 56), (2, 76), (3, 84), (4, 76), (5, 80), (6, 88)]
>>>print("输出元素 76 的下标")
>>>print([i for i,x in b if x==76])                  #利用循环方法获取相应的匹配结果
[2, 4]
```

列表方法如表 3-13 所示。

表 3-13　列表方法

函　　数	描　　述
alist.append(obj)	列表末尾增加元素 obj
alist.count(obj)	统计元素 obj 出现次数
alist.extend(sequence)	用 sequence 扩展列表
alist.index(obj)	返回列表中元素 obj 的索引
alist.insert(index,obj)	在 index 索引之前添加元素 obj
alist.pop(index)	删除索引的元素
alist.remove(obj)	删除指定元素

3.8　元组


元组

3.8.1　概念

元组(Tuple)和列表类似,相当于只读列表,其元素不可以修改。元组适合于只需进行遍历操作的运算,对于数据进行“写保护”,其操作速度比列表快。

元组不可以被修改,代码如下。

```
>>>b=(4,5,6)
>>>b[0]
4
>>>b[0]=1
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    b[0]=1
TypeError: 'tuple' object does not support item assignment
```

如果对于已知的列表 a 进行复制,命名为变量 b,那么 b=a 是无效。此时 b 仅仅是 a 的别名(引用),修改 b 也会修改 a,正确的复制方法应该是 b=a[:]。

```
>>>a=[1,2,3]
>>>b=a
>>>b[0]=4
>>>a
[4, 2, 3]
```

元组与列表相比,具有如下不同点。

(1) 元组在定义时所有元素是放在一对圆括号中,而不是方括号。

(2) 不能向元组增加元素,元组没有 append()、insert()或 extend()方法。

(3) 不能从元组中删除元素,元组没有 remove()或 pop()方法。

(4) 元组没有 index()方法,但可以使用 in()方法。

(5) 元组可以在字典中被用作“键”,但列表不行。

3.8.2 操作

下面介绍元组操作。

1. 创建元组

使用赋值运算符“=”将一个元组赋值给变量,即可创建元组对象。

```
>>>tup1 =('a', 'b', 1997, 2000)
>>>tup2 =(1, 2, 3, 4, 5, 6, 7 )
```

当创建只包含一个元素的元组时,需要注意它的特殊性。此时,只把元素放在圆括号里是不行的,这是因为圆括号既可以表示元组,又可以表示数学公式中的小括号,从而产生歧义。因此,Python 规定:当创建只包含一个元素的元组时,需在元素的后面加一个逗号“,”。

```
>>>x=(1)
>>>x
1
>>>y=(1,)
>>>y
(1,)
>>>z=(1,2)
>>>z
(1, 2)
```

2. 访问元组

可以使用下标索引来访问元组中的值。

```
>>>tup1 =('a', 'b', 1997, 2000)
>>>tup2 =(1, 2, 3, 4, 5, 6, 7 )
>>>print("tup1[0]: ", tup1[0] )
tup1[0]: a
>>>print("tup2[1:5]: ", tup2[1:5] )
tup2[1:5]: (2, 3, 4, 5)
```

3. 元组连接

元组可以进行连接操作。

```
>>>tup1 =(12, 34.56)
>>>tup2 =('abc', 'xyz')
#tup1[0] =100                                  #修改元组元素操作是非法的
>>>tup3 =tup1 +tup2;                           #创建一个新的元组
>>>print(tup3)
(12, 34.56, 'abc', 'xyz')
```

4. 删除元组

元组中的元素值是不允许删除的,但可以使用 del 语句删除整个元组。

```
>>>tup =('physics', 'chemistry', 1997, 2000)
```

```
>>>del tup[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' objext doesn't support item deletion
>>>del tup
>>>print(tup)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'tup' is not defined
```

3.9　字典


字典

3.9.1　字典的概念

【例 3-14】　根据同学的名字查找对应的成绩。

采用列表实现，需要 names 和 scores 两个列表，并且列表中元素的次序一一对应，例如，zhou→95，Bob→75 等，如下。

```
names =['zhou', 'Bob', 'Tracy']
scores =[95, 75, 85]
```

通过名字查找对应成绩，先在 names 中遍历找到所需查找的名字，再从 scores 中遍历取出对应的成绩，查找时间随着列表的长度增加。为了解决这个问题，Python 提供了字典。

字典在其他程序设计语言中称为映射(map)，通过键/值对(key/value)存储数据，键和值之间用冒号间隔，元素项之间用逗号间隔，整体用一对大括号括起来。字典语法结构如下。

```
dict_name={key:value,key:value}
```

字典具有如下特性。

(1) 字典的值可以是任意数据类型，包括字符串、整数、对象，甚至字典。

(2) 键/值对用冒号分隔，而元素项之间用逗号分隔，所有这些都包括在大括号中。

(3) 键/值没有顺序。

(4) 键必须是唯一的，不允许同一个键重复出现，如果同一个键被赋值两次，后一个值会覆盖前面的值。

```
>>>dict ={'Name': 'Zara', 'Age': 7, 'Name': 'Zhou'}
dict['Name']: Zhou
```

(5) 键必须不可变，只能使用数字、字符串或元组充当，不能使用列表，如图 3-11 所示。

```
>>> dict = {['Name']: 'Zhou', 'Age': 7}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

图 3-11　键的取值

因为 dict 根据 key 来计算 value 的存储位置,如果每次计算相同的 key 得出的结果不同,那 dict 内部就完全混乱了。这个通过 key 计算位置的算法称为哈希(Hash)算法。为了保证 Hash 的正确性,作为 key 的对象就不能变。在 Python 中,字符串、整数等都是不可变的,而 list 是可变的,因此,list 不能作为 key。

字典与列表比较,具有以下几个特点。

(1) 字典通过用空间来换取时间,其查找和插入的速度极快,不会随着“键”的增加而增加。

(2) 字典需要占用大量的内存,内存浪费多。

(3) 字典是无序的对象集合,字典当中的元素是通过键来存取的,而不是通过偏移存取。

采用字典实现例 3-14,则只需创建“名字”/“成绩”的键值对,便可直接通过名字查找成绩。字典实现代码如下。

```
>>>d ={'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>d['zhou']
95
```

3.9.2 字典操作

下面介绍字典元素的创建、访问、删除、修改、增加等相关操作。

1. 字典的创建

(1) 使用“=”将一个字典赋给一个变量。

```
>>>a_dict={'Alice':95,'Beth':82,'Tom':65.5,'Emily':95}
>>>a_dict
{'Emily': 95, 'Tom': 65.5, 'Alice': 95, 'Beth': 82}
>>>b_dict={}
>>>b_dict
{}
```

(2) 使用内建函数 dict()。

```
>>>c_dict=dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>>c_dict
{'three': 3, 'one': 1, 'two': 2}
>>>d_dict =dict(one =1, two =2, three =3)
>>>e_dict=dict([('one', 1),('two',2),('three',3)])
>>>f_dict=dict((('one', 1),('two',2),('three',3)))
>>>g_dict=dict()
>>>g_dict
{}
```

(3) 使用内建函数 fromkeys()。

```
>>>h_dict={}.fromkeys((1,2,3),'student')
>>>h_dict
```