第 2 章

Python 语言基础

本章要点

(1) Python 的语法和句法。
(2) Python 的数据类型。
(3) Python 的常量与变量。
(4) Python 的运算符与优先级。
(5) Python 的数值类型。
(6) Python 的字符串类型。
(7) Python 的高级数据类型(列表、元组、字典、集合)。
(8) 正则表达式。

学习目标

(1) 了解 Python 的语法和句法。
(2) 理解 Python 的数据类型。
(3) 掌握 Python 的常量与变量。
(4) 掌握 Python 的数值、字符串、列表、元组、字典、集合。
(5) 掌握正则表达式的概念及其应用。

本章将以较大的篇幅介绍 Python 语言最基础的内容，包括 Python 语法、基本数据类型、常量变量、运算符与优先级、高级数据类型，最后介绍正则表达式的概念及其应用。

2.1 基础 Python 语法

2.1.1 标识符

在编程语言中，标识符就是程序员自己规定的具有特定含义的词，比如类名称、属性名、变量名、函数名等。一般语言规定，标识符由字母或下划线开头，后面可以跟字母、数字、下划线。

Python 标识符的命名规则如下。

(1) 标识符长度无限制。
(2) 标识符不能与关键字(见附录 A)同名。
(3) 字母大小写敏感。
(4) 在 2.x 版本的 Python 中，标识符的命名规则与一般语言的规定一样，但在 3.x 的 Python 中进行了扩展，标识符的引导字符可以是字母、下划线以及大多数非英文语言的字母，只要是 Unicode 编码的字母均可，后续字符可以是上述任意字符，也可以是数字。

虽然 Python 对标识符命名的限制很少，但使用时，仍需要注意以下约定。

(1) 不要使用 Python 预定义的某些标识符，因此要避免使用诸如 NotImplemented 等名字，这些在未来有可能被 Python 的新版本使用。
(2) 不要使用 Python 内建函数名或内置数据类型或异常名作为标识符的名字。
(3) 不要在名字的开头和结尾都使用下划线，因为 Python 中大量地采用这种名字定义各种特殊的方法和变量。

2.1.2　Python 语法和句法

Python 语句中有一些基本规则和特殊字符。

(1)　井号(#)表示其后的字符为 Python 语句的注释。

(2)　换行(\n)是标准的行分隔符(通常一个语句占一行)。

(3)　反斜线(\)继续上一行。

(4)　分号(;)将两条语句放在一行中。

(5)　冒号(:)将复合语句的头和体分开。

(6)　代码块(语句块)用缩进的方式体现。

(7)　用不同的缩进深度分隔不同的代码块。

(8)　Python 文件以模块的形式组织。

1. 注释(#)

尽管 Python 是可读性最好的语言之一，但这并不意味着代码中的注释可以不要。Python 的注释语句以#字符开始，注释可以在一行的任何地方开始，解释器将会忽略该行#之后的所有内容。

2. 续行(\)

一般来讲，Python 的相邻语句使用换行(回车)分隔，亦即一行一条语句。如果一行语句过长，可以使用续行符(\)分解为多行，例如：

```
print("This line is tooooooooo \
long")
```

关于续行符有两种例外情况。

(1)　一个语句不使用反斜线也可以跨行书写。

在使用闭合操作符时，单一语句也可以跨多行。例如，在含有小括号、中括号、花括号时，可以多行书写：

```
print("This is a multiline",
     "example")
```

但须注意，这时的缩进(即使是自动的缩进)将失去语法上的作用。

(2)　三引号内包含的字符串也可以跨行书写。例如：

```
print('''hi there, this is a long message for you
that goes over multiple lines!''')
```

如果要在使用反斜线换行和使用括号元素换行之间做一个选择的话，我们推荐使用后者，因为这样可读性会更好。

3. 多个语句构成代码组(:)

缩进位置相同的一组语句形成一个语句块，亦称代码块或代码组。像 if、for、while、def 和 class 之类的复合语句，首行均以关键字开始，并以冒号(:)结束，该行之后的一行或多行代码就构成了代码组，即语句块。

4. 代码组以不同的缩进分隔

Python 使用缩进来分隔代码组。代码的层次关系是通过相同深度的空格或制表符缩进来体现的，同一代码组内的代码行左边必须严格对齐。换言之，一个代码组内的各行代码，左边必须有数目相同的空格或数目相同的制表符，而且不能以一个制表符替代多个空格！如果不严格遵守这一规则，同一组的代码就可能被视为另一个组，轻则导致逻辑错误，重则导致语法错误。

注意： 对初次使用空白字符作为代码块分界的人来说，首先遇到的问题是：缩进几个空格或制表符才算合适？理论上讲是没有限制的，但我们推荐使用 4 个空格。需要说明一点，不同的文本编辑器中制表符代表的空白宽度不一样，如果所写的代码要跨平台应用，或者将来要被不同的编辑器来读写，那么建议不要使用制表符。

随着缩进深度的增加，代码块的层次也在逐步加深，未缩进的代码块处于最高层次，称作脚本的 main 部分。

采用缩进对齐方式来组织代码，不但代码风格优雅，而且其可读性也大大增强。不仅如此，这种方法还有效地避免了“悬挂 else”(dangling-else)问题，同时也避免了未写大括号时的单一子句问题。试想，如果 C 语言的 if 语句后漏写大括号，而后面却跟着两个缩进的语句，结果会如何呢？毫无疑问，无论条件表达式是否成立，第二个语句总会被执行。这种问题很难调试，不知困惑了多少程序员。

5. 同一行书写多个语句(;)

Python 允许将多个语句写在同一行上，语句之间用分号隔开，而这些语句也不能在这行开始一个新的代码块。例如：

```
a=10; b=20; print(a+b)
```

但必须指出，同一行上书写多个语句，会使代码的可读性大大降低。Python 虽然允许这样做，但并不提倡这么做。

6. 模块

每个 Python 脚本文件均可视为一个模块，它以磁盘文件的形式存在。如果一个模块规模过大，包含的功能太多，就应该考虑对该模块进行拆分，即拆出一些代码另外组建一个或多个模块。模块里的代码既可以是一段直接执行的脚本，也可以是一堆类似库函数的代码，从而可以被别的模块导入(import)后调用。模块可以包含直接运行的代码块、类定义、函数定义，或它们的组合。

2.2 数　　值

2.2.1 数据类型

与 C 等编译型语言不同，Python 中的变量无须声明。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。

在 Python 中，变量就是变量，它没有类型，我们所说的“类型”是变量所指的内存中对象的类型。

赋值运算符(=)用来给变量赋值。

赋值运算符左边必须是一个变量名，右边是存储在变量中的值。例如：

```
>>> counter = 100    # 整型变量
>>> miles = 1000.0   # 浮点型变量
>>> name = "python"  # 字符串
>>> print (counter); print (miles); print (name)
100
1000.0
python
>>>
```

Python 允许同时为多个变量赋值，例如：

a = b = c = 1

注意：

- 以上赋值语句创建了一个整型对象，值为 1，三个变量被分配到相同的内存空间上。以下的运行结果可以证实这一点：

```
>>> a=b=c=1
>>> print("a的地址:",id(a))
a的地址: 1693176272
>>> print("b的地址:",id(b))
b的地址: 1693176272
>>> print("c的地址:",id(c))
c的地址: 1693176272
>>>
```

- Python 中的一切事物皆为对象，并且规定参数的传递都是对象的引用，因此，对象的赋值实际上是对象的引用。

也可以为多个对象指定多个变量，例如：

```
a, b, c = 1, 2, "python"
```

以上形式的赋值语句也称作复合赋值语句，两个整型对象 1 和 2 分配给变量 a 和 b，字符串对象“python”分配给变量 c。

Python 3 中有 6 种标准的数据类型，它们是 Number(数值)、String(字符串)、List(列表)、Tuple(元组)、Dictionary(字典)、Sets(集合)。本节介绍数值类型，后续各节分别介绍其余的各种类型。

Python 3 支持 4 种数值类型：int(整型)、float(浮点型)、bool(布尔型)、complex(复数型)。

在 Python 3 里，只有一种整数类型 int，表示为长整型，而不像 Python 2 那样区分标准整型与长整型。

与大多数编程语言一样，数值类型的赋值和计算都是很直观的。内建的函数 type()可以用来查询变量所指的对象类型。例如：

```
>>> a, b, c, d = 10, 3.5, True, 2+4j
>>> print(type(a), type(b), type(c), type(d))
<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
>>>
```

注意： 在 Python 2 中没有布尔型，它像 C 语言那样，用数字 0 表示 False，用 1 表示 True。在 Python 3 中，把 True 和 False 定义成关键字了，但它们的值仍然是 1 和 0，它们可以和数字参与运算。

当指定一个值时，就创建了一个 Number 对象：

```
var1 = 1
var2 = 2
```

同样，这里的 var1 和 var2 是两个对象引用。使用 del 语句可以删除对象引用。del 语句的语法是：

```
del var1[,var2[,var3[...,varN]]]
```

例如：

```
del var
del var_a, var_b
```

一个变量可以通过赋值指向不同类型的对象。例如：

```
>>> a=10
>>> print(type(a))
<class 'int'>
>>> a="10"
>>> print(type(a))
<class 'str'>
>>>
```

1. 数值运算

像其他语言一样，Python 的数值运算包括加、减、乘、除四则运算以及取余、乘方运算等。例如：

```
>>> 5 + 4              # 加法
9
>>> 4.3 - 2            # 减法
2.3
>>> 3 * 7              # 乘法
21
>>> 2 / 4              # 除法，得到一个浮点数
0.5
>>> 2 // 4             # 除法，得到一个整数
0
>>> 17 % 3             # 取余
2
>>> 2 ** 5             # 乘方
32
>>>
```

注意：

- 除法运算符“ /” 总是返回一个浮点数，要返回整数应使用“ //” 运算符。
- 在混合运算时，Python 把整型数转换成为浮点数。
- 除上述运算外，Python 的整数还支持“ 位运算” ，详见表 2-1。

2. 数值类型实例

Python 3 支持的数值类型实例如下。

int 型实例：10、100、-786、080、-0490、-0x260、0x69。

float 型实例：0.0、15.20、-21.9、32.3e+18、-90.、-32.54e100、70.2e-12。

bool 型实例：True、False。

Python 还支持复数。复数由实部和虚部构成，可以表示为 a + bj 或者 complex(a,b)，其中实部 a 和虚部 b 都是浮点型。

complex 型实例：3.14j、45.j、9.322e-36j、.876j、-.6545+0j、3e+26j、4.53e-7j。

2.2.2　常量与变量

几乎所有的编程语言都有变量和常量的概念，它们与数学上的概念很相似。

1. 变量

变量与数学函数中的变量一样。顾名思义，变量就是其值可以改变的量，只不过在程序中变量不仅仅可以是数字，还可以是字符串等。变量名的命名方式严格遵循上一节提及的 Python 标识符命名规则，即由英文字母、Unicode 字符(含汉字)、数字、下划线组成，且不能以数字开头。

下面的例子通过赋值运算符把 1 赋值给 a，这个 a 就是我们创建的一个变量：

```
>>> a=1
>>> a
1
>>>
```

又如：

```
a=1
b=2
```

这里我们用一个字母来表示变量，通过赋值运算符来给变量赋值。

Python 是动态语言，所以我们不需要像 C 语言那样提前声明一个变量的数据类型。C 语言使用变量之前需要声明一下。例如：

```
int a=1;
```

而 Python 直接使用 a=1 即可。这就是动态语言的好处，我们无须关心变量本身的数据类型，Python 有自己的判断机制。

注意：给变量赋值的时候，Python 只会记住最后一次所赋的值。例如：

```
>>> a=1
>>> a=2
>>> a
2
>>>
```

先给 a 赋值 1，再赋值 2，最后输出 a 的时候显示的是最后一次所赋的值 2。

2. 常量

常量与变量相对应，就是程序运行过程中其值不可改变的量。例如：

```
PI=3.1415926
```

这时，我们就认定 PI 是个常量，也就是说，PI 始终代表 3.1415926。实际上，Python 中并没有严格意义的常量，因为 Python 中没有保证常量不会改变的机制。一般来讲，我们使用全部大写的字母来表示常量。尽管我们称 PI 为常量，但仍然可以给 PI 重新赋值。

换言之，Python 的世界里本来就没有常量，编程时主动不修改的变量也就伪装成了常量。用大写字母来“注明”常量，只有提醒的意义，其实这个值还是可以改变的。

在编程过程中，我们可能会用到常量的概念，所以这里还是要提一下。

2.2.3 运算符与优先级

1. Python 的运算符

Python 的运算符十分丰富，表 2-1 简要列示了 Python 的运算符及其用法。

表 2-1 Python 运算符及其用法

运算符	名 称	说 明	例 子
+	加	两个对象相加(对字符串则是连接)	3 + 5 得到 8。 ‘a’ + ‘b’ 得到 ‘ab’
-	减/负号	得到一个负数，或是一个数减去另一个数	-5.2 得到一个负数。 50 - 24 得到 26
*	乘	两个数相乘或是返回一个被重复若干次的字符串	2*3 得到 6。 ‘la’*3 得到 ‘lalala’
**	幂	指数运算，返回 x 的 y 次幂	3**4 得到 81 (即 3*3*3*3)
/	除	x 除以 y	4/3 得到 1.3333333333333333
//	取商数	返回商的整数部分	4 // 3.0 得到 1.0
%	取余数	返回除法的余数	8%3 得到 2。 -25.5%2.25 得到 1.5
<<	左移	把一个数的位向左移一定数目(每个数在内存中都表示为位或二进制数字，即 0 和 1)	2 << 1 得到 4。因为 2 用二进制表示时为 10B，往左移一位变成二进制的 100B，二进制的 100B 用十进制表示则为 4

续表

运算符	名　称	说　明	例　子
>>	右移	把一个数的位向右移一定数目	11>>1 得到 5。因为 11 用二进制表示时为 1011B，向右移动 1 位后得到 101B，即十进制的 5
&	位和 (Bitwise AND)	数的位和	5 & 3 得到 1。 (101B &11B 得到 1B)
|	位或 (Bitwise OR)	数的位或	5 | 3 得到 7。 (101B | 11B 得到 111B)
~	位翻转	位翻转	x 的位翻转是-(x+1)，~5(101B)得到-6 (-110B)
<	小于	比较运算符，返回 x 是否小于 y 的结果。返回 1 则表示真，返回 0 则表示假。Python 中也可以用特殊的变量 True 表示真，用 False 表示假。注意首字母为大写	5<3 返回 0(即 False)；而 3<5 返回 1(即 True)。也可以任意连接比较运算符：3 < 5 < 7 返回 True
>	大于	返回 x 是否大于 y	5>3 返回 True。如果两个操作数都是数值，它们会先被转换成相同的类型后才开始计算。如果 x 和 y 类型不同，则会返回 False
<=	小于或等于	返回 x 是否小于或等于 y	x = 3; y = 6; x <= y 返回 True
>=	大于或等于	返回 x 是否大于或等于 y	x = 4; y = 3; x >= y 返回 True
==	等于	比较两个对象是否相等	x = 2; y = 2; x == y 返回 True。 x = ‘abc’; y = ‘Abc’; x == y 返回 False
!=	不等于	比较两个对象是否不相等	x = 2; y = 3; x != y 返回 True
not	布尔非	如果 x 为 None(空)、0、False 或空字符串，则 not x 会返回 True，否则 not x 会返回 False	x = True; not x 返回 False。 x = False; not x 返回 True
and	布尔与	如果 x 为 False，x and y 返回 False，否则它会返回 y 的计算值	x = False; y = True; x and y 的结果会返回 False。在这个例子里，Python 语言并不会计算 y 的值，因为 x 的值已经是 False，所以这个表达式的值肯定是 False。这个现象称为短路计算
or	布尔或	如果 x 为 True，则返回 True，否则返回 y 的计算值	x = True; y = False; x or y 的结果会返回 True。短路计算在这里也同样适用

2. Python 运算符的优先级

表 2-2 给出 Python 运算符的优先级，其排列顺序是从最低的优先级(最松散地结合)到最高的优先级(最紧密地结合)。这意味着在计算一个表达式时，Python 首先计算列在表下部的运算符，然后再计算列在表上部的运算符。

表 2-2 Python 运算符的优先级

运 算 符	描 述
lambda	Lambda 表达式
or	布尔或
and	布尔与
not x	布尔非
in、not in	成员测试
is、is not	同一性测试
<、<=、>、>=、!=、==	比较
|	按位或
^	按位异或
&	按位与
<<、>>	移位
+、-	加法与减法
*、/、%	乘法、除法、取余数
+x、-x	正负号
~x	按位翻转
**	指数
x.attribute	属性参考
x[index]	下标
x[index:index]	寻址段
f(arguments...)	函数调用
(expression,...)	绑定或元组显示
[expression,...]	列表显示
{key:datum,...}	字典显示
‘expression,...’	字符串转换

上述运算符优先级表决定了哪个运算符在别的运算符之前计算。然而，如果想要改变它们的计算顺序，可以使用圆括号。例如，想要在一个表达式中让加法在乘法之前计算，那么就要写成类似于(1 + 2) * 3 的形式。

3. Python 运算符的结合规律

运算符通常自左向右结合，即具有相同优先级的运算符按照从左向右的顺序计算。例

如，1 + 2 + 3 被解释为(1 + 2) + 3。也有一些运算符是自右向左结合的，如赋值运算符。这样的运算符，其结合顺序正相反，即 a = b = c 被解释为 a = (b = c)。

注意： 合理使用括号能够增强代码的可读性。在很多场合下，使用括号都是一个好主意，而没用括号的话，可能会使程序得到错误的结果，或使代码的可读性降低，引起阅读者的困惑。括号在 Python 语言中不是必须存在的，不过为了获得良好的可读性，使用括号总是值得的。

2.3 字　符　串

1. Python 字符串

字符串(string)是 Python 中最常用的数据类型，Python 使用引号(单引号 ' 或双引号 ")作为字符串的定界符(一般为单行字符串，多行字符串使用三引号，稍后介绍)。

要创建一个字符串，只要用成对的引号把若干个字符括起来即可，例如：

```
var1 = 'Hello World!'
```

```
var2 = "Python Runoob"
```

Python 规定，单引号内可以使用双引号，这时双引号被视为一个普通字符，不再作为定界符，反之亦然。例如：

```
str1='I want a book,\n and you want a book,too.'    #单引号
str2='"I want a book,\n and you want a book,too."'  #单引号中使用双引号
str3="I want a book,\n and you want a book,too."    #双引号
str4="'I want a book,\n and you want a book,too.'"  #双引号中使用单引号
print(str1); print(str2); print(str3); print(str4)
```

```
I want a book,
 and you want a book,too.
"I want a book,
 and you want a book,too."
I want a book,
 and you want a book,too.
'I want a book,
 and you want a book,too.'
>>>
```

一个字符串用什么引号开头，就必须用什么引号结尾，首尾定界符不匹配时将出错，例如：

```
>>> str="Wrong string'
SyntaxError: EOL while scanning string literal
>>>
```

2. 字符串中值的访问(子串截取)

与 C 语言不同，Python 不支持字符类型，即使是单个字符，Python 也将其视为一个字符串来使用。

访问子串，实际上就是截取字符串中的子串，有的文献称其为“切片运算”，使用的运算符是方括号([]或[:]或[::])，例如：

```
str="0123456789"
print ("str[0:3]:",str[0:3]) #截取第一位到第三位的字符
print ("str[:]:",str[:]) #截取字符串的全部字符
print ("str[6:]:",str[6:]) #截取第七个字符到结尾
print ("str[:-3]:",str[:-3]) #截取从头开始到倒数第三个字符之前
print ("str[2]:",str[2]) #截取第三个字符
print ("str[-1]:",str[-1]) #截取倒数第一个字符
print ("str[::-1]:",str[::-1]) #创建一个与原字符串顺序相反的字符串
print ("str[-3:-1]:",str[-3:-1]) #截取倒数第三位与倒数第一位之前的字符
print ("str[-3:]: ",str[-3:]) #截取倒数第三位到结尾
print ("str[:-5:-3]:",str[:-5:-3]) #逆序截取
```

上述程序段的运行结果如下：

```
str[0:3]: 012
str[:]: 0123456789
str[6:]: 6789
str[:-3]: 0123456
str[2]: 2
str[-1]: 9
str[::-1]: 9876543210
str[-3:-1]: 78
str[-3:]:  789
str[:-5:-3]: 96
>>>
```

从上面的例子可以看出：①要获得单个字符，应使用切片运算符 s[i]，但 i 不能省略；②要获得一个连续的子串，应使用切片运算符 s[i:j]，它返回字符串 s 中从索引 i(包括 i)到 j(不包括 j)之间的子串，若 i 被省略，Python 就认为 i=0，若 j 被省略，Python 就认为 j=len(s)-1，其中 len(s)表示字符串的长度(稍后介绍)；③要每间隔若干字符返回一个字符，应使用切片运算符 s[i:j:k]，k 为间隔字符的个数，k 为正数表示从左向右截取，k 为负数表示从右向左截取，k 不能为 0。不难想象，s[::-1]获得的将是逆序的整个字符串。

3. Python 字符串的更新

不能企图通过切片运算更新已存在的字符串，如：

```
>>> str='0123456789'
>>> str[0]='a'
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    str[0]='a'
TypeError: 'str' object does not support item assignment
>>>
```

必须使用 Python 的内建函数 replace()实现字符串的更新，例如：

```
>>> str.replace('0','a')
'a123456789'
>>>
```

4. Python 字符串中的转义字符

当字符串中需要使用特殊字符时，可以像 C 语言那样，在特殊字符前冠以反斜杠(\)，形成转义字符。Python 的转义字符如表 2-3 所示。

表 2-3　Python 的转义字符

转义字符	描　述
\(在行尾时)	续行符
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\0yy	八进制数 yy 代表的字符，例如：\012 代表换行
\xyy	十六进制数 yy 代表的字符，例如：\x0a 代表换行
\other	其他的字符以普通格式输出

5. Python 的字符串运算符

假如 a 变量的值为字符串“Hello”，而 b 变量的值为字符串“Python”，那么，各种运算结果如表 2-4 所示。

表 2-4　Python 的字符串运算符

操作符	描　述	实　例
+	字符串连接	a+b 输出结果：HelloPython
*	重复输出字符串	a*2 输出结果：HelloHello
[]	通过索引获取字符串中的字符	a[1] 输出结果：e
[:]	截取字符串中的一部分(切片)	a[1:4] 输出结果：ell
in	成员运算符——如果字符串中包含给定的字符，则返回 True	‘H’ in a 输出结果：1
not in	成员运算符——如果字符串中不包含给定的字符，则返回 True	‘M’ not in a 输出结果：1
r/R	原始字符串——所有的字符串都是直接按照字面的意思来使用，没有转义为特殊的或不能打印的字符。原始字符串除在字符串的第一个引号前加上字母 r(可以大小写)以外，与普通字符串有着几乎完全相同的语法	print (r‘\n’) 输出结果：\n print (R‘\n’)输出结果：\n
%	格式化字符串	见后文叙述

下面将上述字符串运算放入一个程序中：

```
#!/usr/bin/python3
a = "Hello"
b = "Python"
print("a + b 输出结果: ", a + b)
print("a * 2 输出结果: ", a * 2)
print("a[1] 输出结果: ", a[1])
print("a[1:4] 输出结果: ", a[1:4])
if("H" in a) :
   print("H 在变量 a 中")
else :
   print("H 不在变量 a 中")
if("M" not in a) :
   print("M 不在变量 a 中")
else :
   print("M 在变量 a 中")
print(r'\n')
print(R'\n')
```

以上程序的输出结果如下：

```
>>>
a + b 输出结果： HelloPython
a * 2 输出结果： HelloHello
a[1] 输出结果： e
a[1:4] 输出结果： ell
H 在变量 a 中
M 不在变量 a 中
\n
\n
>>>
```

6. Python 字符串的格式化

Python 支持字符串的格式化输出。尽管这样做可能会使表达式非常复杂，但最基本的用法是将一个值插入到一个有字符串格式符 %s 的字符串中。

在 Python 中，字符串格式化的语法与 C 中 printf 函数相似。我们看下面的例子：

```
>>> print ("我叫 %s,今年 %d 岁!" % ('张三', 20))
我叫 张三,今年 20 岁!
>>>
```

Python 字符串格式化用到的符号如表 2-5 所示。

表 2-5　Python 字符串格式化用到的符号

符　号	描　述
%c	格式化字符及其 ASCII 码
%s	格式化字符串
%d	格式化整型数
%u	格式化无符号整型数

续表

符　号	描　述
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数(大写)
%f	格式化浮点数，可指定小数点后的精度
%e	用科学计数法格式化浮点数
%E	作用同%e，用科学计数法格式化浮点数
%g	按%f 和%e 的较短者输出
%G	按%f 和%E 的较短者输出
%p	用十六进制数格式化变量的地址

格式化操作符前面可以加入辅助操作符，这一点也与 C 语言极为相似。Python 的格式化辅助操作符如表 2-6 所示。

表 2-6　Python 格式化辅助操作符

符　号	功　能
*	定义宽度或者精度
-	输出左对齐
+	在正数前面显示加号(+)
<sp>	在正数前面显示空格
#	在八进制数前面显示零('0')，在十六进制数前面显示'0x'或者'0X'(取决于用的是'x'还是'X')
0	显示的数字前面填充'0'而不是默认的空格
%	'%%'输出一个单一的'%'
(var)	映射变量(字典参数)
m.n.	m 是显示的最小总宽度，n 是小数点后的位数(如果可用的话)

尽管使用格式化操作符输出字符串很方便，而且与 C 语言兼容，但 Python 并不推荐使用这种方法，具体原因参见本书 4.1.2 小节。

7. Python 的三引号

Python 的三引号(三个单引号或三个双引号均可)用于字符串跨多行，字符串中可以包含换行符、制表符以及其他特殊字符。举例如下：

```
>>> para_str = """这是Python多行字符串的实例
多行字符串可以使用制表符
TAB ( \t )。
也可以使用换行符 [ \n ]。
"""

>>> print (para_str)
这是Python多行字符串的实例
多行字符串可以使用制表符
TAB (    )。
也可以使用换行符 [
 ]。

>>>
```

三引号把程序员从引号和特殊字符串的泥潭里解脱出来，自始至终保持一小块字符串的格式，符合所见即所得(What You See Is What You Get，WYSIWYG)的特征。

一个典型的应用是，当需要一段 HTML 或者 SQL 时，如果用字符串组合转义字符，将会非常繁琐，而使用三引号不失为一种绝佳的选择，例如：

```
errHTML = '''
<HTML><HEAD><TITLE>
Friends CGI Demo</TITLE></HEAD>
<BODY><H3>ERROR</H3>
<B>%s</B><P>
<FORM><INPUT TYPE=button VALUE=Back
ONCLICK="window.history.back()"></FORM>
</BODY></HTML>
'''
cursor.execute('''
CREATE TABLE users (
login VARCHAR(8),
uid INTEGER,
prid INTEGER)
''')
```

三引号中可以使用成对出现的字符串定界符，例如：

```
str5='''I want a book,and you want a book,\n too.'''    #三单引号
str6='''"I want a book,and you want a book,\n too."'''
  #三单引号中间使用双引号

str7='''I want a book,
and you want a book,\n too.'''   #三单引号中有换行符
str8="""I want a book,
and you want a book,\n too."""   #三双引号中有换行符
print(str5); print(str6); print(str7); print(str8)
```

上面程序段的运行结果如下：

```
I want a book,and you want a book,
 too.
"I want a book,and you want a book,
 too."
I want a book,
and you want a book,
 too.
I want a book,
and you want a book,
 too.
```

注意： Python 字符串的几种定界符均已介绍完毕，现对其归纳如下。

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。

④ 单引号和双引号不能搭配使用，必须成对使用。

8. Unicode 字符串

在 Python 2 中，普通字符串是以 8 位 ASCII 码形式进行存储的，而 Unicode 字符串则存储为 16 位 Unicode 字符串。使用 Unicode 字符串，旨在表示更多的字符，其语法非常简单，只要在字符串前面加上前缀 u 即可。

Python 3 中默认所有的字符串都为 Unicode 的，因而不必在字符串的前面加前缀 u。

9. 原始字符串

前面讲解字符串运算符时已提及原始字符串并举例(表 2-4)，此处再强调一下。在字符串前面加字母 r 或 R，就是告诉 Python 解释器，其后的字符串是原始字符串(Raw String)。原始字符串指的是，字符串内的所有字符均保持其原有的含义，不做转义处理。换言之，“\n”在原始串中是两个字符，即“\”和“n”，而不会被转义为换行符。由于正则表达式(本章最后介绍)经常与“\”冲突，所以，当一个字符串中使用了正则表达式之后，往往在前面加字母 r，具体例子参见 2.8.2 节。此外，在描述文件路径时，往往使用“\”，为避免歧义，也往往在其前面冠以字母 r，具体例子见 5.1 节。

10. Python 的字符串内建函数

Python 的字符串有很多内建函数，常用的如表 2-7 所示。

表 2-7　常用的 Python 字符串内建函数

序　号	方法及描述
1	capitalize() 将字符串的第一个字符转换为大写
2	center(width, fillchar) 返回一个指定的宽度 width 居中的字符串，fillchar 为填充的字符，默认为空格
3	count(str, beg=0, end=len(string)) 返回 str 在 string 里面出现的次数，如果指定 beg 或 end，则返回指定范围内 str 出现的次数
4	bytes.decode(encoding=“utf-8”, errors=“strict”) Python 3 中没有 decode 方法，但我们可以使用 bytes 对象的 decode()方法来解码给定的 bytes 对象，这个 bytes 对象可以由 str.encode()来编码返回
5	encode(encoding=‘utf-8’, errors=‘strict’) 以 encoding 指定的编码格式编码字符串，如果出错，则默认报一个 ValueError 的异常，除非 errors 指定的是‘ignore’或者‘replace’
6	endswith(suffix, beg=0, end=len(string)) 检查字符串是否以 suffix 结束，如果指定 beg 或者 end，则检查指定的范围内是否以 suffix 结束，如果是，则返回 True，否则返回 False
7	expandtabs(tabsize=8) 把字符串中的 Tab 符号转为空格，Tab 符号默认的空格数是 8
8	find(str, beg=0, end=len(string)) 检测 str 是否包含在字符串中，如果用 beg 和 end 指定范围，则检查是否包含在指定的范围内，如果是，则返回开始的索引值，否则返回-1

续表

序　号	方法及描述
9	index(str, beg=0, end=len(string)) 跟 find()方法一样，只不过如果 str 不在字符串中，则会报一个异常
10	isalnum() 如果字符串至少有一个字符，并且所有字符都是字母或数字，则返回 True，否则返回 False
11	isalpha() 如果字符串至少有一个字符，并且所有字符都是字母，则返回 True，否则返回 False
12	isdigit() 如果字符串只包含数字，则返回 True，否则返回 False
13	islower() 如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False
14	isnumeric() 如果字符串中只包含数字字符，则返回 True，否则返回 False
15	isspace() 如果字符串中只包含空格，则返回 True，否则返回 False
16	istitle() 如果字符串是标题化的(见 title())，则返回 True，否则返回 False
17	isupper() 如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True，否则返回 False
18	join(seq) 以指定字符串作为分隔符，将 seq 中所有的元素(字符串表示)合并为一个新的字符串
19	len(string) 返回字符串长度，即字符串中字符的个数
20	ljust(width[, fillchar]) 返回一个原字符串左对齐，并使用 fillchar 填充至长度 width 的新字符串，fillchar 默认为空格
21	lower() 转换字符串中所有大写字母为小写
22	lstrip() 截掉字符串左边的空格
23	maketrans(intab, outtab) 创建字符映射的转换表，对于接受两个参数的最简单的调用方式，第一个参数是字符串，表示需要转换的字符，第二个参数也是字符串，表示转换的目标
24	max(str) 返回字符串 str 中最大的字母
25	min(str) 返回字符串 str 中最小的字母

续表

序　号	方法及描述
26	replace(old, new [, max]) 将字符串中的 old 替换成 new。如果 max 指定，则替换不超过 max 次
27	rfind(str, beg=0, end=len(string)) 类似于 find()函数，不过是从右边开始查找
28	rindex(str, beg=0, end=len(string)) 类似于 index()，不过是从右边开始
29	rjust(width [, fillchar]) 返回一个原字符串右对齐，并使用 fillchar(默认空格)填充至长度 width 的新字符串
30	rstrip() 删除字符串末尾的空格
31	split(str= “”, num=string.count(str)) num=string.count(str))以 str 为分隔符截取字符串，如果 num 有指定值，则仅截取 num 个子字符串
32	splitlines([keepends]) 按照行(‘\r’, ‘\r\n’, ‘\n’)分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符
33	startswith(str, beg=0, end=len(string)) 检查字符串是否是以 str 开头，是则返回 True，否则返回 False。如果 beg 和 end 指定值，则在指定范围内检查
34	strip([chars]) 在字符串上执行 lstrip()和 rstrip()
35	swapcase() 将字符串中大写转换为小写，小写转换为大写
36	title() 返回“标题化”的字符串，就是说，所有单词都是以大写开始，其余字母均为小写(见 istitle())
37	translate(table, deletechars=“”) 根据 table 给出的表(包含 256 个字符)转换 string 的字符, 需要过滤掉的字符放到 deletechars 参数中
38	upper() 转换字符串中的小写字母为大写
39	zfill(width) 返回长度为 width 的字符串，原字符串右对齐，前面填充 0
40	isdecimal() 检查字符串是否只包含十进制字符，如果是则返回 True，否则返回 False

这些内建函数为我们处理字符串带来了极大的方便。读者学习 Python 语言时，不要试图自己编写程序来处理字符串，应当充分利用这些内建函数完成相应的功能。

细心的读者可能已经注意到，表 2-7 的标题上写的是“函数”，而表格栏目中谓之“方法”。其实，函数与方法并无严格的区别。在面向过程的语言中，一个模块主要强调的是数据处理，就像数学上的函数一样，故称为函数；在面向对象的语言中，一般把类中

定义的函数称作方法、服务或操作，因为它主要强调这个类的对象封装了一些属性和方法(变量和函数)并向外提供服务。表 2-7 中所列函数均是定义在某个类里面的，故称为方法。说到底，方法就是类函数。

有鉴于此，在不产生混淆的前提下，本书的后续章节拟不严格区分函数与方法。

有关类与面向对象的概念，读者可参阅第 6 章中的相关内容。

下面看几个关于字符串操作的例子(其中使用了下一节将要介绍的列表)：

```
>>> myString = "+"
>>> seq = ['1', '2','3']
>>> myString.join(seq)
'1+2+3'
>>> myString = "This is an apple and that is an apple too."
>>> myString.replace('apple', 'orange', myString.count('apple'))
'This is an orange and that is an orange too.'
>>> myString.replace('apple', 'orange', 1)
'This is an orange and that is an apple too.'
>>> myString.split(' ', 3)
['This', 'is', 'an', 'apple and that is an apple too.']
>>>
```

2.4 列表与序列

在介绍列表之前，我们先了解一下 Python 中“序列”的概念。在 Python 中，序列是最基本的数据结构。序列中的每个元素都被分配一个数字，以表明它的位置，并称为索引。其中，第一个索引值为 0，第二个索引值为 1，依此类推。

Python 中的序列都可以进行索引、切片、加、乘、检查成员等操作。为了使用方便，Python 还内建了确定序列长度以及确定最大和最小元素的方法。

Python 中最常用的序列是列表和元组，本节介绍列表，元组放在下一节介绍。

1. 列表的概念

列表(list)是 Python 中使用最频繁的数据类型，它是放在方括号[]内、用逗号分隔的一系列元素。

列表中，元素的类型可以不同，支持数字、字符串，甚至可以包含列表。换言之，列表允许嵌套。

2. 列表的创建

创建一个列表时，只要把逗号分隔的不同的数据项用方括号括起来即可。例如：

```
list1 = ['Google', 'Runoob', 1997, 2017]
list2 = [1, 2, 3, 4, 5 ]
list3 = ["a", "b", "c", "d"]
```

与字符串一样，列表同样可以被索引、截取和组合。列表被截取后，返回一个包含所需元素的新列表。与字符串的索引一样，列表索引也是从 0 开始的。

3. 列表的访问与截取

可以使用下标索引来访问列表中的元素，同样，也可以使用类似于字符串切片运算的

形式截取列表中的元素，例如：

```
>>> list1 = ['Google', 'Runoob', 1997, 2017]
>>> list2 = [1, 2, 3, 4, 5, 6, 7 ]
>>> print ("list1[0]: ", list1[0])
list1[0]:  Google
>>> print ("list2[1:5]: ", list2[1:5])
list2[1:5]:  [2, 3, 4, 5]
>>>
```

令：

```
L=['Google', 'Runoob', 'Taobao']
```

则截取操作如表 2-8 所示。

表 2-8　Python 列表的截取操作

Python 表达式	结　果	描　述
L[2]	‘Taobao’	读取第三个元素
L[-2]	‘Runoob’	从右侧开始读取倒数第二个元素
L[1:]	[‘Runoob’, ‘Taobao’]	输出从第二个元素开始的所有元素

下面将上述列表截取操作放入一个程序中：

```
#!/usr/bin/python3
L=['Google', 'Runoob', 'Taobao']
print(L[2])
print(L[-2])
print(L[1:])
```

以上程序的输出结果如下：

```
Taobao
Runoob
['Runoob', 'Taobao']
>>>
```

注意： 字符串、列表和元组(下节介绍)三者都是序列，都支持切片运算，操作方法也极为相似，只是操作结果的类型有所不同。此外，不能通过切片运算对字符串和元组进行更新，列表则可以。

4. 列表的更新

可以对列表的数据项进行修改或更新，也可以使用 append()方法(稍后介绍)添加一些列表项。例如：

```
>>> list = ['Google', 'Runoob', 1997, 2017]
>>> print ("第三个元素为 : ", list[2])
第三个元素为 :  1997
>>> list[2] = 2001
>>> print ("更新后的第三个元素为 : ", list[2])
更新后的第三个元素为 :  2001
>>>
```

5. 列表元素的删除

可以使用 del 语句来删除列表中的元素，例如：

```
>>> list = ['Google', 'Runoob', 1997, 2017]
>>> print (list)
['Google', 'Runoob', 1997, 2017]
>>> del list[2]
>>> print ("删除第三个元素后 : ", list)
删除第三个元素后 :  ['Google', 'Runoob', 2017]
>>>
```

使用 remove()方法也可以删除列表的元素，我们稍后再讨论。

6. 列表操作符

列表对 + 和 * 的操作符与字符串相似。+ 号用于组合列表，* 号用于重复列表。+ 和 * 的用法如表 2-9 所示。

表 2-9　列表中+和*的用法

Python 表达式	结　果	描　述
len([1, 2, 3])	3	长度(即列表中元素的个数)
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	组合(即拼接)
[‘Hi!’] * 4	[‘Hi!’, ‘Hi!’, ‘Hi!’, ‘Hi!’]	重复
3 in [1, 2, 3]	True	元素是否存在于列表中
for x in [1, 2, 3]: print (x,end=‘ ’)	1 2 3	遍历并输出列表的各个元素

注意： 为了让列表的各个元素打印在一行上，而且相邻两个元素之间隔一个空格，表 2-9 的最后一行使用了 print (x,end=‘ ’)这种打印格式，后面遍历元组和集合时采用了同样的方法，3.4.6 节对此将做详解。

7. 列表嵌套

列表嵌套指的是在列表里创建其他列表，例如：

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
>>>
```

8. Python 列表中的内建函数与方法

Python 列表中的内建函数如表 2-10 所示。

表 2-10　Python 列表中的内建函数

序　号	函　数
1	len(list)　返回列表元素的个数
2	max(list)　返回列表元素的最大值
3	min(list)　返回列表元素的最小值
4	list(seq)　将元组、字典、集合、字符串等转换为列表

Python 列表中的方法如表 2-11 所示。

表 2-11　Python 列表中的方法

序　号	方　法
1	list.append(obj)　在列表末尾添加新的对象
2	list.count(obj)　统计某个元素在列表中出现的次数
3	list.extend(seq)　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	list.index(obj)　从列表中找出某个值第一个匹配项的索引位置
5	list.insert(index, obj)　将对象插入列表
6	list.pop(obj=list[-1])　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
7	list.remove(obj)　移除列表中某个值的第一个匹配项
8	list.reverse()　将列表中的元素反向
9	list.sort([func])　对原列表进行排序
10	list.clear()　清空列表
11	list.copy()　复制列表

下面看几个列表操作的例子：

```
>>> #向列表尾部添加元素
>>> a=[1,2,3,4]
>>> a.append(5)
>>> print(a)
[1, 2, 3, 4, 5]
>>>
```

```
>>> #插入一个元素
>>> a=[1,2,4]
>>> a.insert(2,3)
>>> print(a)
[1, 2, 3, 4]
>>>
```

```
>>> #扩展列表
>>> a=[1,2,3]
>>> b=[4,5,6]
>>> a.extend(b)
>>> print(a)
[1, 2, 3, 4, 5, 6]
>>>
```

```
>>> #删除元素（如有重复元素，只会删除最靠前的）
>>> a=[1,2,3,2]
>>> a.remove(2)
>>> print(a)
[1, 3, 2]
>>>
```

```
>>> #删除指定位置的元素，默认为最后一个元素
>>> a=[1, 2, 3, 4, 5, 6]
>>> a.pop()
6
>>> print(a)
[1, 2, 3, 4, 5]
>>> a.pop(3)
4
>>> print(a)
[1, 2, 3, 5]
>>>
>>> #逆序
>>> a=[1, 2, 3, 4, 5, 6]
>>> a.reverse()
>>> print(a)
[6, 5, 4, 3, 2, 1]
>>>
>>> #排序
>>> a=[2,4,7,6,3,1,5]
>>> a.sort()
>>> print(a)
[1, 2, 3, 4, 5, 6, 7]
>>>
```

2.5 元　　组

Python 的元组(tuple)与列表相似，不同之处在于元组的元素是不能修改的。另外，列表使用方括号[]，而元组使用圆括号()。

1. 元组的创建

元组的创建很简单，只需要在括号中添加元素，并使用逗号分隔各元素即可。例如：

```
tup1 = ('Google', 'Runoob', 1997, 2017)
tup2 = (1, 2, 3, 4, 5 )
tup3 = ("a", "b", "c", "d")
```

创建空元组时，使用如下语句：

```
tup1 = ()
```

注意：

- 当元组中只包含一个元素时，需要在元素后面添加逗号，例如 tup1 = (50,)。
- 与字符串类似，元组的下标索引也从 0 开始，而且也可以进行截取、组合等操作。

2. 元组的访问

可以使用下标索引来访问元组中的元素，例如：

```
>>> tup1 = ('Google', 'Runoob', 1997, 2017)
>>> tup2 = (1, 2, 3, 4, 5, 6, 7 )
>>> print ("tup1[0]: ", tup1[0])
tup1[0]:  Google
>>> print ("tup2[1:5]: ", tup2[1:5])
tup2[1:5]:  (2, 3, 4, 5)
>>>
```

3. 元组的修改

如前所述，元组中的元素值是不允许修改的，但可以对元组进行连接组合。假如：

```
>>> tup1 = (12, 34.56)
>>> tup2 = ('abc', 'xyz')
```

以下修改元组元素的操作是非法的：

```
>>> tup1[0] = 100
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    tup1[0] = 100
TypeError: 'tuple' object does not support item assignment
>>>
```

但可以通过连接运算符“+”创建一个新的元组：

```
>>> tup3 = tup1 + tup2
>>> print (tup3)
(12, 34.56, 'abc', 'xyz')
>>>
```

4. 元组的删除

既然元组不能修改，那么其中的元素值也就不允许删除，但我们可以使用 del 语句删除整个元组，例如：

```
>>> tup = ('Google', 'Runoob', 1997, 2017)
>>> print (tup)
('Google', 'Runoob', 1997, 2017)
>>> del tup
>>> print (tup)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    print (tup)
NameError: name 'tup' is not defined
>>>
```

可见，元组被删除后，再输出该元组时会显示异常信息。

5. 元组运算符

与字符串类似，元组之间可以使用 + 号和 * 号进行运算。这就意味着可以将它们组合和复制，运算后将生成一个新的元组。

表 2-12 给出了几个例子。

表 2-12　Python 元组的运算符

Python 表达式	结　果	描　述
len((1, 2, 3))	3	计算元素个数
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	连接
(‘Hi!’,) * 4	(‘Hi!’, ‘Hi!’, ‘Hi!’, ‘Hi!’)	重复
3 in (1, 2, 3)	True	元素是否存在于元组中
for x in (1, 2, 3): print (x,end=‘ ’)	1 2 3	遍历并输出元组的各个元素

6. 元组的索引、截取

前已提及，元组是一个序列，所以可以访问元组中指定位置的元素，也可以通过索引截取元组中的一段元素。假设元组为 L = (‘Google’, ‘Taobao’, ‘Runoob’)，表 2-13 说明了元组的索引和截取。

表 2-13　Python 元组的索引和截取

Python 表达式	结　果	描　述
L[2]	‘Runoob!’	读取第三个元素
L[-2]	‘Taobao’	反向读取；读取倒数第二个元素
L[1:]	(‘Taobao’, ‘Runoob!’)	截取元素，从第二个开始后的所有元素

运行结果如下：

```
>>> L = ('Google', 'Taobao', 'Runoob')
>>> L[2]
'Runoob'
>>> L[-2]
'Taobao'
>>> L[1:]
('Taobao', 'Runoob')
>>>
```

7. 元组的内建函数

Python 元组包含了一些内建函数，如表 2-14 所示。

表 2-14　Python 元组的内建函数

序号	方法及描述	举　例

1	len(tuple)
	计算元组中元素的个数
		`>>> tuple1 = ('Google', 'Runoob', 'Taobao')`
		`>>> len(tuple1)`
		`3`
		`>>>`

2	max(tuple)
	返回元组中元素的最大值
		`>>> tuple2 = ('5', '4', '8')`
		`>>> max(tuple2)`
		`'8'`
		`>>>`

续表

序号	方法及描述	举　例

3	min(tuple)
	返回元组中元素的最小值
		>>> tuple2 = ('5', '4', '8')
		>>> min(tuple2)
		'4'
		>>>

4	tuple(seq)
	将列表、字符串、字典、集合等转换为元组
		>>> list1= ['Google', 'Taobao', 'Runoob', 'Baidu']
		>>> tuple1=tuple(list1)
		>>> tuple1
		('Google', 'Taobao', 'Runoob', 'Baidu')
		>>>

2.6　字　　典

1. 字典的概念

Python 中的字典(dictionary)是一种可变容器模型，且可以存储任意类型的对象。

字典中的每个项都是“键/值对”(有时写作“键-值对”或“键值对”)，“键”与“值”之间用冒号(:)分隔，而每个“对”之间用逗号(,)分隔，整个字典放在花括号{}中，格式如下：

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

对每个键/值对而言，键必须是唯一的，但值可以改变。值可以取任何数据类型，但键必须是不可变的。例如，字符串、数字或元组均可作为键，但列表不可以。

2. 字典的创建

Python 中创建字典的方法很简单，只要将键值对放入花括号内，并用逗号隔开即可。一个简单的字典例子如下：

```
dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
```

也可这样创建字典：

```
dict1 = { 'abc': 456 }
dict2 = { 'abc': 123, 98.6: 37 }
```

3. 字典的访问

把相应的键放入方括号内，即可得到相应的值。例如：

```
>>> dict1 = {'Name':'Runoob','Age':7,'Class':'First'}
>>> print("dict1['Name']",dict1['Name'])
dict1['Name'] Runoob
>>> print("dict1['Age']",dict1['Age'])
dict1['Age'] 7
>>>
```

如果所用的键在字典中不存在，则数据无法访问，会输出错误信息，例如：

```
>>> dict1 = {'Name':'Runoob','Age':7,'Class':'First'}
>>> print("dict1['Alice']",dict1['Alice'])

Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    print("dict1['Alice']",dict1['Alice'])
KeyError: 'Alice'
>>>
```

4. 字典的添加与修改

向字典添加新的键/值对，便向字典中添加了新的内容。修改或添加已有键/值对的例子如下：

```
>>> dict1 = {'Name':'Runoob','Age':7,'Class':'First'}
>>> dict1['Age'] = 8                    #更新age
>>> dict1['School'] = "TJPU"            #添加信息
>>> print("dict1['Age']:",dict1['Age'])
dict1['Age']: 8
>>> print("dict1['School']:",dict1['School'])
dict1['School']: TJPU
>>>
```

5. 字典元素的删除

既能删除字典中的单一元素，也能将整个字典清空，而且清空只需一项操作。

删除一个字典用 del 命令，例如：

```
>>> dict1 = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}
>>> del dict1['Name'] # 删除键 'Name'
>>> dict1
{'Class': 'First', 'Age': 7}
>>> dict1.clear()     # 删除字典元素
>>> dict1
{}
>>> print ("dict1['Age']: ", dict1['Age'])
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    print ("dict1['Age']: ", dict1['Age'])
KeyError: 'Age'
>>> del dict1         # 删除字典
>>> print ("dict1['Age']: ", dict1['Age'])
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    print ("dict1['Age']: ", dict1['Age'])
NameError: name 'dict1' is not defined
>>> dict1
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    dict1
NameError: name 'dict1' is not defined
>>>
```

第一次引发异常是因为字典中已无元素(空字典)，试图访问键‘age’会出错；第二次引发异常是因为前面已执行了 del 操作，字典已不复存在，当然不能访问键‘age’；第三次引发异常也是因为字典不存在，所以两次引发异常的名称均为 NameError。

6. 字典键的特性

字典的值可以取任何 Python 对象，没有任何限制，它既可以是标准对象，也可以是用户自己定义的对象，但键不行。

关于字典的键，有两点必须牢记。

(1) 同一个键不得出现两次。创建字典时，如果同一个键被赋值两次，则后一个值被记住，例如：

```
>>> dict1 = {'Name':'Runoob','Age':7,'Name':'小菜鸟'}
>>> print("dict1['Name']",dict1['Name'])
dict1['Name'] 小菜鸟
>>>
```

(2) 键必须不可变。可以用数字、字符串或元组做键，用列表则不行，例如：

```
>>> dict1 = {['Name']:'Runoob','Age':7}
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    dict1 = {['Name']:'Runoob','Age':7}
TypeError: unhashable type: 'list'
>>>
```

7. 字典的内建函数与方法

Python 字典包含的内建函数如表 2-15 所示。

表 2-15　Python 字典的内建函数

序号	函数及描述	举　例

1	len(dict)
	计算字典元素个数，即键的总数
		`>>> dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}`
		`>>> len(dict)`
		`3`

2	str(dict)
	输出字典，以可打印的字符串表示
		`>>> dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}`
		`>>> str(dict)`
		`"{'Name': 'Runoob', 'Class': 'First', 'Age': 7}"`

3	type(variable)
	返回输入的变量类型，如果变量是字典就返回字典类型
		`>>> dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}`
		`>>> type(dict)`
		`<class 'dict'>`

Python 字典包含的内建方法如表 2-16 所示。

表 2-16　Python 字典的内建方法

序号	函数及描述
1	radiansdict.clear() 删除字典内所有元素
2	radiansdict.copy() 返回一个字典的浅拷贝
3	radiansdict.fromkeys(seq[, val]) 创建一个新字典，以序列 seq 中的元素做字典的键，val 为字典所有键对应的初始值(缺省时为 None)

续表

序号	函数及描述
4	radiansdict.get(key, default=None) 返回指定键的值，如果值不在字典中，则返回 default 值
5	key in dict 如果键在字典 dict 里，则返回 True，否则返回 False
6	radiansdict.items() 以列表返回可遍历的(键, 值)元组数组
7	radiansdict.keys() 以列表返回一个字典所有的键
8	radiansdict.setdefault(key, default=None) 与 get()类似，但如果键不存在于字典中，将会添加键并将值设为 default
9	radiansdict.update(dict2) 把字典 dict2 的键/值对更新到字典里
10	radiansdict.values() 以列表返回字典中的所有值

下面看几个典型的例子：

```
>>> #返回一个具有相同键-值对的新字典（浅复制）
>>> x={'name':'admin','machines':['foo','bar','bax']}
>>> y=x.copy()
>>> y['name']='ylh' #替换值，原字典不受影响
>>> y['machines'].remove('bar') #修改了某个值(原地修改不是替换)，原字典会改变
>>> y
{'name': 'ylh', 'machines': ['foo', 'bax']}
>>> x
{'name': 'admin', 'machines': ['foo', 'bax']}
>>>
>>> #使用给定的键建立新的字典，每个键默认的对应的值为none
>>> {}.fromkeys(['name','sex','age'])
{'name': None, 'age': None, 'sex': None}
>>> dict.fromkeys(['name','sex','age'])
{'name': None, 'age': None, 'sex': None}
>>> dict.fromkeys(['name','sex','age'],'(unknown)')
{'name': '(unknown)', 'age': '(unknown)', 'sex': '(unknown)'}
>>>
>>> #访问指定键的值
>>> d={}
>>> print (d['name'])  #键不存在，出错
Traceback (most recent call last):
  File "<pyshell#72>", line 1, in <module>
    print (d['name'])  #键不存在，出错
KeyError: 'name'
>>> print (d.get('name'))
None
>>> d.get('name','N/A')
'N/A'
>>> d['name']='Eric'
>>> d.get('name')
'Eric'
>>>
```

```
>>> #利用一个字典更新另外一个字典
>>> d={'x':1,'y':2,'z':3}
>>> f={'y':5}
>>> d.update(f)
>>> d
{'x': 1, 'z': 3, 'y': 5}
>>>
```

2.7　集　　合

1. 集合概述

集合(set)是 Python 的基本数据类型，把不同的元素组合在一起便形成了集合。组成一个集合的成员称作该集合的元素(element)。在一个集合中不能有相同的元素。下面是集合的例子：

```
>>> li=['a','b','c','a']
>>> se =set(li)
>>> se
{'b', 'c', 'a'}
>>>
```

可以看到，相同的元素被自动删除了。

集合对象是一组无序排列的可哈希的值，集合成员可以作为字典的键。相比之下，列表对象是不可哈希的，所以下面的程序会出错：

```
>>> li=[['a','b','c'],['a','c']]
>>> se = set(li)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    se = set(li)
TypeError: unhashable type: 'list'
>>>
```

集合可以分为两类：可变集合(set)与不可变集合(frozenset)。

可变集合可添加和删除元素，是非可哈希的，不能用作字典的键，也不能做其他集合的元素。而不可变集合与之相反。

2. 集合操作符和关系符号

集合有各种操作，各种操作符和关系符号如表 2-17 所示。

表 2-17　集合操作符和关系符号

数学符号	Python 符号	说　明
∈	in	是……的成员
∉	not in	不是……的成员
=	==	等于
≠	!=	不等于
⊂	<	是……的真子集
⊆	<=	是……的子集