《Python程序设计基础教程（微课版）》

http://dblab.xmu.edu.cn/post/python

第5章 字符串

林子雨 博士/副教授

厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

主页：http://dblab.xmu.edu.cn/linziyu

主讲教师

2017年度厦门大学奖教金获得者
2020年度厦门大学奖教金获得者

主讲教师：厦门大学 林子雨 博士/副教授
中国高校首个“**数字教师**”提出者和建设者
2009年7月从事教师职业以来
累计**免费**网络发布超过1500万字高价值教学和科研资料
网络浏览量超过1500万次

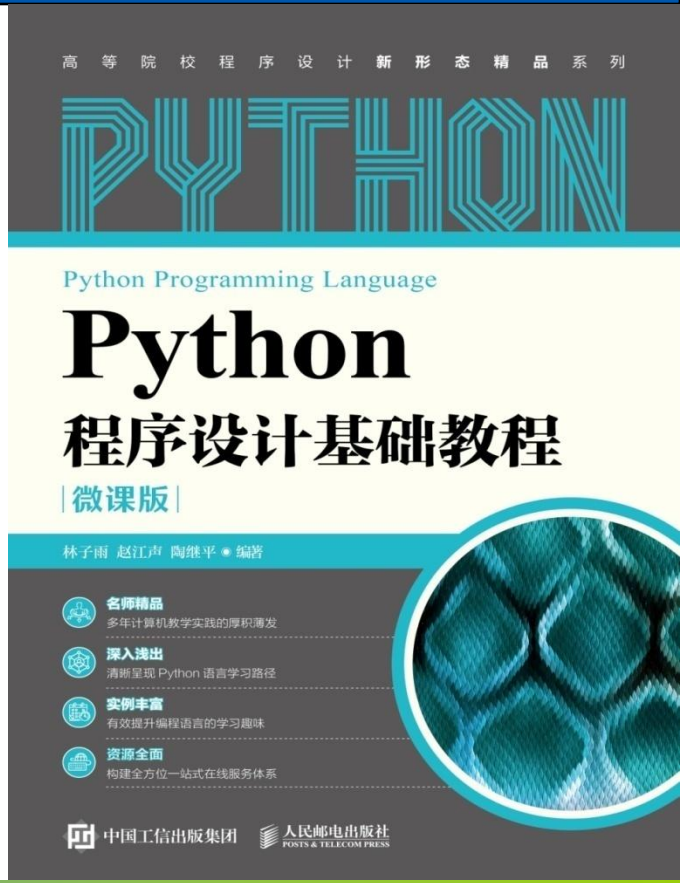
提纲

5.1 基本概念
5.2 字符串的索引和切片
5.3 字符串的拼接
5.4特殊字符和字符转义
5.5原始字符串和格式化字符串
5.6字符串的编码
5.7 常用操作

本PPT是如下教材的配套讲义：
《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《 Python程序设计基础教程（微课版） 》教材官方网站：
http://dblab.xmu.edu.cn/post/python



5.1 基本概念

字符串是Python的六大数据结构之一，这种类型的数据可谓是最常使用的数据类型了，在使用print函数的过程中，它的第一个参数就是一个字符串。字符串是一个不可变类型，即声明之后，其中的所有字符都不能再修改。判定一个变量是否是字符串，需要使用isinstance函数，例如：

```
>>> testString = "ts"
>>> isinstance(testString, str)
True
```

可以看到，该函数有两个参数，第一个参数是需要判断的变量本身，第二个参数是str，也就是字符串（string）的缩写。当返回值为True时，说明该值是一个字符串；当返回值为False时，该值不是一个字符串。这个函数经常用于判断输入的值的类型是否被正确地转换。比如，用户输入了“2021-05-01”，在经过某个函数转换后，需要判断它是一个日期还是一个字符串时，就需要使用isinstance函数。

5.1 基本概念

字符串的声明非常简单，将一串字符使用单引号或者双引号包裹起来就可以了。在Python中，无论哪种引号包裹起来都是字符串。但需要注意的是，引号必须配对使用，比如，不能以双引号开始、单引号结束。下面是字符串的一些实例：

```
>>> aString = 'Hello World'
>>> bString = "I'm a String"
>>> cString = '<div class="my"></div>'
>>> dString = "这是一个错误的示例，不能双引号开始，单引号结束'
SyntaxError: EOL while scanning string literal
```

5.1 基本概念

被引号包裹的内容称为“字符串字面量”，在有具体语境的环境下，也可以直接称为“字面量”或者“字面值”。比如，上面的aString的字面量就是Hello World。字面量加包裹用的引号合在一起才是“字符串”。比如，aString的字符串是”Hello World”（包含引号）。当然，在书写时，不用管声明用的是单引号还是双引号，只要写出一对正确的引号就可以。字符串开始和结束的引号必须配对使用，但是，在字符串中间就不需要配对使用了。

可以看到，上述过程中，选用的字符串都只有一行。而在实际操作中，字符串经常需要编写多行。比如，一段Python的源代码就是一个多行字符串。在Python中，可以使用三重引号来表示一个多行字符串，这种表达法称为“长字符串”。具体实例如下：

```
>>> aString = '''\
#-*- coding:utf-8 -*-
x = 1;
y = 1;
print(x + y);
'''
>>> print(aString)
#-*- coding:utf-8 -*-
x = 1;
y = 1;
print(x + y);
```

在这里，三重引号同样需要配对使用。实际上，多行注释的本质就是一个没有被赋值给变量的多行字符串，因为它没有被赋值给其它变量，所以解释器在解释的时候就将其直接跳过，也就达到了多行注释的目的。

在上面的实例中还可以看到，第一行结尾有一个“\”。这个符号是一个转义符，意味着本行结尾的换行符不计入输出之中。如果不加这个符号，输出效果如下：

```
>>> aString = '''
#-*- coding:utf-8 -*-
x = 1;
y = 1;
print(x + y);
'''
>>> print(aString)
# 注意，这里会输出一个空行
#-*- coding:utf-8 -*-
x = 1;
y = 1;
print(x + y);
```

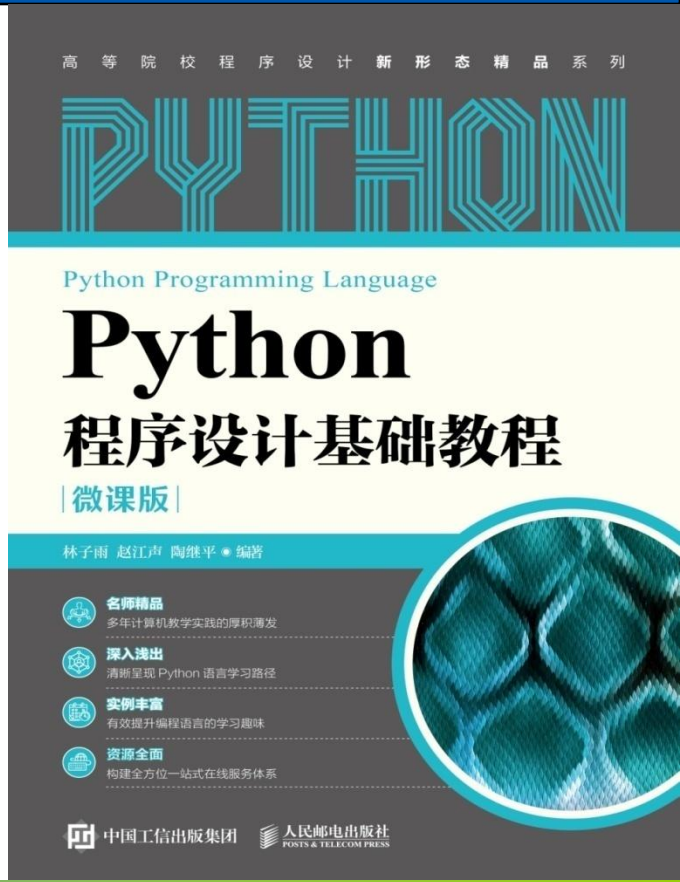
注意观察上面的“print(aString)”的输出，多出了一个空行。所以，如果当一个多行字符串结束时，如果不希望在输出时将换行符也输出，就需要在行尾增加一个“\”。在实际使用中，这种写法可以让输出更加美观。比如，下面的写法就不是很美观：

```
>>> aString = '''#-*- coding:utf-8 -*-
x = 1;
y = 1;
print(x + y);
'''
```

所以，适当地使用“\”可以让代码变得更加美观。

5.2 字符串的索引和切片

5.2.1 字符串的索引
5.2.2 字符串的切片



本PPT是如下教材的配套讲义：
《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《 Python程序设计基础教程（微课版） 》教材官方网站：
http://dblab.xmu.edu.cn/post/python

5.2.1 字符串的索引

字符串的本质是字符的组合，在一个字符串中，其每一个组成部分都称为一个字符。图5-1给出了一个包含5个字符的字符串，其中，每个字符所在的位置称为“字符的偏移量”，通过偏移量来查询字符串中指定位置字符的方法称为“索引查询”。当然，在实际操作过程中，如果说“偏移量为2的字符是一个感叹号”会显得很奇怪，所以一般直接说“索引值为2的字符是一个感叹号”就可以了。也就是说，在实际的使用环境下，直接说“索引”指的就是偏移量。

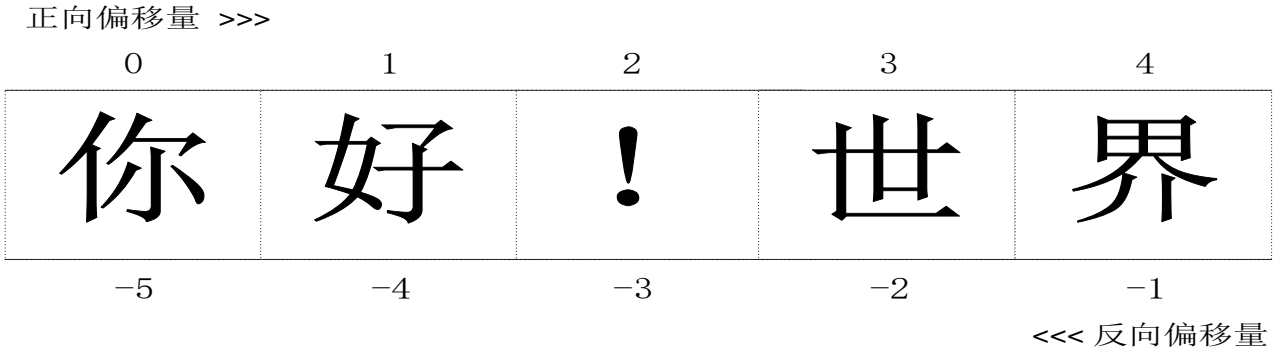


图5-1 字符的偏移量

5.2.1 字符串的索引

字符串的索引可以分为两种：正向索引和反向索引。其中，正向索引指的是符合阅读习惯的索引方式，比如，我们平时阅读中文时，按照从左至右的顺序阅读。也就是说，正向索引的开始点是字符串的左边第一个字符，其索引值是0，之后每个字符的索引值依次增加1。也就是说，正向索引中所有的索引值都是正数或0。

反向索引则是反其道而行之，按照从右至左的顺序编号。由于索引值0已经被正向索引使用，为了不产生歧义，反向索引从-1开始，从右至左依次减1。也就是说，反向索引所有的索引值都是负数。

5.2.1 字符串的索引

在字符串中，使用下标运算符“[]”来查询指定索引值对应的字符，具体实例如下：

```
>>> aString = "你好！世界"
>>> aString[3]
'世'
>>> aString[-4]
'好'
>>> aString[-0]
'你'
```

5.2.1 字符串的索引

在使用索引时，需要特别注意以下两点：
（1）数学上-0、+0和0都指的是同一个数字0，所以，哪怕使用的是-0为索引值，所代表的也是正向索引的第一个字符。反向索引的第一个字符的索引值是-1；
（2）索引的结果是一个只读的值。与C、Java、C#等语言相比，由于Python的基础数据类型里没有字符型，所以，不能通过改变索引对应值的方法修改字符串，否则会报错，下面是一个具体实例：

```
>>> aString = "你好！世界"
>>> aString[2] = '2'
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    aString[2] = '2'
TypeError: 'str' object does not support item assignment
```

5.2.1 字符串的索引

在Python中，修改字符串都是通过字符串拼接的方法实现的。比如，在上面这个实例中，代码试图修改字符串的第3个字符，那么就需要取出第1和2个字符，再取出第4和5个字符，然后将这些字符与修改后的字符进行拼接，再赋予给原来的变量，以达到修改的效果。总体而言，当需要修改字符串索引值为n的字符时，基本步骤如下：

（1）求出字符串的长度m；
（2）如果m<n，那么返回失败；
（3）取出索引值为0至n-1的字符串，记为leftString；
（4）取出索引值为n+1至m-1的字符串，记为rightString；
（5）把leftString、修改后的字符和rightString拼接为一个新的字符串newString；
（6）输出newString。

5.2.2 字符串的切片

字符串的切片操作与列表的对应操作类似，不同点在于返回的是一个字符串而不是列表。由于返回值是原字符串的一部分，所以这里也可以将返回值称为原字符串的“子字符串”，或者简称为“子串”。切片操作的基本方法如下：

（1）返回[m,n)的子串，可以使用aString[m:n]这种写法。这里的m必须小于n，同时，返回的值包含m而不包含n。比如，原字符串aString="string"，aString[1:3]的值为"tr"，也就是"string"中索引值为1和2对应的字符串。m和n也可以是负数，但是需要注意负数的大小关系，不能写成aString[-1:-3]，第一个参数值必须小于第二个参数值，正确的写法是aString[-3:-1]。同样地，这种情况下的返回值是不包括-1所代表的字符的。

5.2.2 字符串的切片

如果使用了错误的索引值，那么系统将返回一个空字符串，而不会提示一个错误或者异常，所以，这是一个无论任何时候都可以安全使用的方法。具体实例如下：

```
>>> aString = "string"
>>> aString[1:3]
'tr'
>>> aString[-3:-1]
'in'
>>> aString[-1:-3]
''
```

5.2.2 字符串的切片

（2）如果m和n分别指的是字符串的开头或者结尾，那么就可以不写。比如，查询从第2个字符开始到字符串结尾的子串，就可以写成aString[1:]。再比如，反向查询从开头到倒数第二个字符，就可以写成aString[:-2]。特别地，如果m和n都不写，那么就代表着字符串从头取到尾，这也是一个特别的子串，也就是字符串本身，写法是aString[:]。具体实例如下：

```
>>> aString = "string"
>>> aString[1:]
'tring'
>>> aString[:-2]
'stri'
>>> aString[:]
'string'
```

5.2.2 字符串的切片

（3）切片的第三种写法是aString[m::n]，用于从字符串的索引值为m（即第m+1个字符）开始，每n个字符取出一次的情况。假设原字符串aString为“string”，下面根据n为正、n为负、不写n以及不写m与n等四种情况分别说明：

①如果n为正，则查询方向为正向索引的方向。例如，aString[1::2]表示的是从索引值为1的字符（即第2个字符）开始，向右每2个字符取一个，也就是索引值为1、3、5的字符组成的字符串“tig”。aString[-5::2]表示的是从索引值为-5的字符开始，向右每2个字符取一个，也就是索引值为-5、-3、-1的字符组成的字符串，也是“tig”。特别地，m不写则代表从索引值0开始查询。

②如果n为负，则查询的方向为反向索引的方向。例如，aString[5::-2]表示的是从索引值为5的字符（即第6个字符）开始，向左每2个取一个字符，也就是索引值为5、3、1的字符组成的字符串"git"。aString[-1::-2]表示的是从索引值为-1的字符开始，向左每2个字符取一个，也就是索引值为-1、-3、-5的字符组成的字符串，也为"git"。特别地，m不写则代表从索引值-1开始查询。

5.2.2 字符串的切片

③如果不写n，则视为aString[m::1]。例如，aString[3::]的值为"ing"，aString[-3::]的值也是"ing"，它们分别是从索引值为3和为-3的字符开始，以正向索引的方向取每个字符直至字符串结尾。

④如果n和m都不写，则视为aString[0::1]，即从字符串的开头开始取每一个字符，显然结果就是字符串本身。特别地，如果n为0，则系统会返回一个错误。

5.2.2 字符串的切片

由此可以发现，aString[m::n]本质为，从索引值为m的字符开始，取出m、m+n、m+2n一直到m+kn为止所代表的字符串。这里k指的是在字符串边界里可以取到的最大正整数。所以，在这个操作中，m称为“起始位置”，n称为“步长”。上述所有操作的具体实例如下：

```
>>> aString = "string"
>>> aString[1::2]
'tig'
```

aString[1::2]表示从索引值为1的字符开始取值，一直取到字符串末尾，并且步长为2，也就是每2个字符取出1个。索引为1的字符是t，步长为2，下一个字符就是i，再下一个字符就是g，所以最终结果是tig。

5.2.2 字符串的切片

```
>>> aString[-5::2]
'tig'
```

aString[-5::2]表示从索引值为-5的字符开始取值，一直取到字符串的末尾，并且步长为2，也就是每2个字符取出1个。索引为-5的字符是t，再往右走，步长为2，下一个字符是i，再下一个字符就是g，所以最终结果是tig。

```
>>> aString[::2]
'srn'
```

aString[::2]表示从头到尾取值，步长为2。所以，取出的第1个字符是s，取出的第2个字符是r，取出的第3个字符是n，所以，最终取出的所有字符是srn。

5.2.2 字符串的切片

```
>>> aString[5::-2]
'git'
```

aString[5::-2]表示从索引值为5的字符开始从右向左移动取值，步长为2，所以，取出的第1个字符是索引为5的字符，也就是g，然后往左侧移动，步长为2，取出的第2个字符是索引值为3的字符，也就是i，然后，再往左侧移动，步长为2，取出的第3个字符是索引值为1的字符，也就是t，最终取出来的字符是git。

```
>>> aString[-1::-2]
'git'
```

aString[-1::-2]表示从索引值为-1的字符开始从右向左移动取值，步长为2.取出的第1个字符是索引值为-1的字符，也就是g，取出的第2个字符是索引值为-3的字符，也就是i，取出的第3个字符是索引值为-5的字符，也就是t，所以，最终取出来的字符是git。

5.2.2 字符串的切片

```
>>> aString[::-2]
'git'
```

aString[::-2]表示从字符串的尾部向头部移动取值，步长为2.取出的第1个字符是索引值为-1的字符，也就是g，取出的第2个字符是索引值为-3的字符，也就是i，取出的第3个字符是索引值为-5的字符，也就是t，所以，最终取出来的字符是git。

```
>>> aString[::0]
Traceback (most recent call last):
  File "<pyshell#44>", line 1, in <module>
    aString[::0]
ValueError: slice step cannot be zero
>>> aString[::]
'string'
```

5.3 字符串的拼接

•在学习了什么是字符串以后，就要开始学习对字符串进行的运算。在Python中，字符串的运算只有两种，即拼接和索引。索引用于查询字符串中某个字符，而拼接用于将数个短的字符串连接成为一个长的字符串。

•字符串拼接主要包括加号连接、%连接、join函数、format函数和格式化字符串等方法。

5.3 字符串的拼接

所谓“加号连接”是指直接将两个字符串使用加号连接在一起。比如，要拼接字符串"thon"和变量prefix = "py"，可以使用如下方式：

```
>>> prefix = "py"
>>> result = prefix + "thon"
>>> result
'python'
```

这种方法简单明了，但是效率不高。从其运行原理来说，由于Python的字符串是不可变类型，因此，一个字符串在生成之后如果要修改，就只能新生成一个新的字符串。对于一个字符串操作“a+b+c+d+e+...”而言，在运行时，系统会按照规则先将a和b拼接为一个新字符串，再将新字符串与c拼接造成一个新字符串，依此类推。也就是说，如果有n个字符串拼接，那么中间就会生成n-2个临时字符串，哪怕这些字符串程序员是看不到的，但是还是一样会消耗内存空间。

为了解决上面这个问题，就需要使用%连接，这种连接方式只需要申请一次内存空间。比如，对于上面的这个实例，如果使用%连接，则采用如下方式：

```
>>> prefix = "py"
>>> result = "%sthon" % prefix
>>> result
'python'
```

如果读者学习过C语言，看到上述代码时应该会感到非常亲切，这个就是C语言中格式字符串的写法。当然，随着Python的不断发展和进步，这种方法也已经在Python 2.6版本中被淘汰，取而代之的是format函数和格式化字符串。

5.3 字符串的拼接

在字符串拼接中，还有一种场景会使用到，那就是重复输出一个字符若干次。比如，要显示一个进度值为40%的滚动条，就需要使用如下方法显示：

```
>>> bar = "\u2589"
>>> print("4/10:[" + bar * 4 + "--" * 6 + "] = 40%")
4/10:[■■■■----------] = 40%
```

这里使用乘号表示一个字符串出现若干次，“\u2589”代表一个黑色的方块。

5.3 字符串的拼接

还有一种非常常用的拼接方法就是join函数，可以将一个列表拼接为一个字符串，这种方式广泛地用于生成参数表，比如生成URL的查询字符串。该函数是字符串的成员函数，所以使用时需要以一个字符串为对象才能使用，指定的字符串将成为拼接的分隔符。比如需要将列表[“Hello”, “World”, “Python”]用逗号拼接，可以采用如下方式：

```
>>> ",".join(["Hello", "World", "Python"])
'Hello,World,Python'
```

5.3 字符串的拼接

【例5-1】假设有一个字典{“username”: “sample”, “code”: “2002001001”, “source”: “python”}，使用字符串拼接方法生成该字典对应的URL查询字符串。所谓的URL查询字符串指的是URL里的QueryString部分，其生成规则是将字典里的每一项写成“key=value”的形式，然后使用“&”进行分隔，即需要写成“username=sample&code=2002001001&source=python”的形式。

程序的编写思路如下：

（1）遍历字典，将字典中的每一项变成“key=value”的形式，存入一个列表中；

（2）以“&”作为分隔符对象，使用join函数，将列表转变为结果要求的字符串。

5.3 字符串的拼接

基于以上编程思路，具体实现代码如下：

```
01 # string_join.py
02 sourceDict = {
03     "username": "sample",
04     "code": "2002001001",
05     "source": "python"
06 }
07
08 array = [];
09 for (key, value) in sourceDict.items():
10     array.append(key + "=" + value)
11
12 queryString = "&".join(array)
13 print(queryString)
```

该程序的执行结果如下：

username=sample&code=2002001001&source=python

5.3 字符串的拼接

从上文可知，字符串拼接是一种运算，比如使用str+str1，那么在运行过程中就有3个字符串字面量，也就是str、str1和它们的运算结果。实际上，在一部分情况下，在编写代码时，字符串的内容就已经确定了，只不过字符串可能特别长，如果写成一个字符串，就会使代码变得非常难以阅读，比如：

```
>>> sql = "SELECT ID, Name, AccessLevel FROM [Users] WHERE
Name=@p1 AND Password=@p2";
```

在书写时，这样一长串字符串就显得非常没有条理、没有章法，并且不好阅读。那么如果将它写成这样：

```
>>> sql = "SELECT ID, Name, AccessLevel"
>>> sql = sql + " FROM [Users]"
>>> sql = sql + " WHERE Name=@p1 AND Password=@p2";
```

5.3 字符串的拼接

虽然看上去，这里只有一个变量sql，但是，运行过程中一共会产生5个字符串字面量。所以，对于这种情况，在Python中，可以直接将多个字符串字面量连续书写，从而将其自动连接：

```
>>> sql = ("SELECT ID, Name, AccessLevel "
        "FROM [Users] "
        "WHWERE Name=@p1 AND Password=@p2")
>>> sql
'SELECT ID, Name, AccessLevel FROM [Users] WHWERE Name=@p1 AND Password=@p2'
```

可以看出，在书写时，直接将两个字符串常量写在一起，就能将其拼接起来。比如"Py'"thon'，就可以得到结果'Python'：

```
>>> "Py'"thon'
'Python'
```

5.3 字符串的拼接

还可以直接使用长字符串的写法，实例如下：

```
>>> sql = """\
SELECT ID, Name, AccessLevel \
FROM [Users] \
WHERE Name=@p1 AND Password=@p2\
"""
>>> sql
'SELECT ID, Name, AccessLevel FROM [Users] WHERE Name=@p1 AND
Password=@p2'
```

可以看出，使用长字符串的方法和使用字符串自动连接的方法，返回的结果完全一致，并且后者看起来更简单明了。此外，在书写源代码时，每一行都写一个“\”并不美观，所以，正常来说都是使用两个字符串拼接的方式来书写。

5.4 特殊字符和字符转义

•在字符串的实际使用过程中，存在着一些无法直接显示的字符，比如换行符、水平制表符等，这些符号被称为“特殊字符”。

•此外，如果在使用双引号包裹的字符串中必须使用双引号，那么这种情况下双引号也是一种“特殊字符”。

•为了在字符串中表达这些字符，就需要使用字符转义的形式来书写。字符转义的方法是使用“\”加一些特定的字符。

5.4 特殊字符和字符转义

在Python中，常用的转义字符如表5-1所示。

表5-1 常用的转义字符

转义字符	含义
\newline	使用时是反斜杠加换行 实际效果是反斜杠加换行全部被忽略
\\	反斜杠(\)
\'	单引号(')
\"	双引号(")
\n	换行符(LF)
\r	回车符(CR)
\t	水平制表符(TAB)
\ooo	表示一个八进制数码位的字符，比如 \141 表示的是字母a
\xhh	表示一个十六进制码位的字符，比如 \x61 表示的是字母a

5.4 特殊字符和字符转义

需要注意的是，标记为“\newline”的转义字符并不是写为“\newline”，实际的使用方法如下：

```
>>> sql = """\
SELECT ID, Name, AccessLevel \
FROM [Users] \
WHERE Name=@p1 AND Password=@p2\
"""
>>> sql
'SELECT ID, Name, AccessLevel FROM [Users] WHERE Name=@p1 AND Password=@p2'
```

在这里，每一行结尾的“\”即是这个转义符，其含义是将反斜杠和之后的换行符全部不显示在实际的字符串中。

5.4 特殊字符和字符转义

•另外两个容易混淆的是转义符“\r”和“\n”。在ASCII码中，CR表示的是回车符，使用的是“\r”。

•所谓回车，指的是老式打字机在一行打印完以后，装纸滚筒移到最右边的动作。

•换行才是指切换至下一行。所以，表现在实际应用中，“\r”指的是将光标位置回到行首，“\n”指的是直接切换至下一行，且光标位置不变。也就是说，在键盘上，按下“Enter”键产生的实际效果应该是“\r\n”。

•因此，我们平时所说的“回车符”，也就是键盘上的回车键按下所产生的特殊符号，其准确命名应该是“回车式换行符”。

不过，在现在的新系统中，人们发现“\r\n”基本上是连用的，已经很少有换行但不回车的操作了。因此，在新版的系统中“\n”就表示“回车式换行符”，而不再需要使用“\r”了。实际效果如下所示：

```
>>> sql = "SELECT * \nFROM [Users] \nWHERE user=@p1"
>>> print(sql)
SELECT *
FROM [Users]
WHERE user=@p1
```

需要特别注意的一点是，在IDLE中，回车符“\r”是不显示的。所以，如果需要使用到“\r”，那么就需要将Python程序放在命令行窗口中运行。

5.4 特殊字符和字符转义

水平制表符（TAB）指的就是键盘上的TAB键。每一个“\t”的实际效果就是点击了一次键盘上的TAB键。在一般的场景中，比如IDLE或者记事本，每一个“\t”表示的就是将光标移到最近的8的倍数个字符的位置上。水平制表符是为了达到对齐的目的，在输出时，前后各加一个“|”字符，就正好是一个10格宽的单元格。由于“垂直制表符”并不常用，所以一般来说，“制表符”就代表着“水平制表符(TAB)”。具体实例如下：

```
>>> table = "xyz\t5字符\tabc\n123456781234567812345678"
>>> print(table)
xyz         5字符       abc
123456781234567812345678
```

5.4 特殊字符和字符转义

之前曾提到过，所有的源代码都可以视为字符串，Python的源代码也不例外。在运行Python代码时，解释器会首先从文件中将源代码以字符串的形式读入内存中，再进行操作。在Python的源代码中，缩进使用的是“空格符”而不是“制表符”。Python的缩进其实是4个空格，并且每一个空格都可以单独选中，所以应该是4个“空格符”。而如果使用其它的语言，比如C#，它们的缩进经常使用的是“制表符”，也就是说，虽然也有4个空格符的宽度，但是不能单独选中每一个空格，所以是“制表符”。

在Python 3.x中，缩进推荐使用“空格符”，并且不允许混合使用“制表符”和“空格符”。如果打开的是Python 2.x的代码，那么就需要将混合使用的缩进统一转换为“空格符”。

5.4 特殊字符和字符转义

最后一类制表符指的是使用八进制或者十六进制码位。这里所谓的码位指的是ASCII码字符的码位，比如，字母'a'在ASCII中的码位是97，在Python中，就可以使用转义的方法，将其表现出来。比如，十进制数48转换为8进制数是60，转换为16进制数是30，因此，ASCII码中48所表示的字符就可以用如下方式来显示：

```
>>> print("\60")
0
>>> print("\x30")
0
```

5.4 特殊字符和字符转义

从表5-1可以看到，有一个转义符是“\ooo”，这里代表着最多只能使用一个3位的八进制数。同理，转义符“\xhh”指的是只能使用2位的十六进制数。比如“回车符”所代表的ASCII码是13，也就是十六进制数D，那么就必须写成“\x0D”，而不能写成“\xD”。

5.4 特殊字符和字符转义

单引号和双引号的转义符虽然不常用，但也有需要注意的内容。首先，无论包裹字符串使用的是哪种引号，将引号转义肯定不会错。其次，如果包裹的引号和字符串中使用的引号不同，那么引号可以不转义。具体实例如下：

```
>>> print("\'")
'
>>> print('\"')
"
>>> print("I'm fine, thank you")
I'm fine, thank you
```

5.4 特殊字符和字符转义

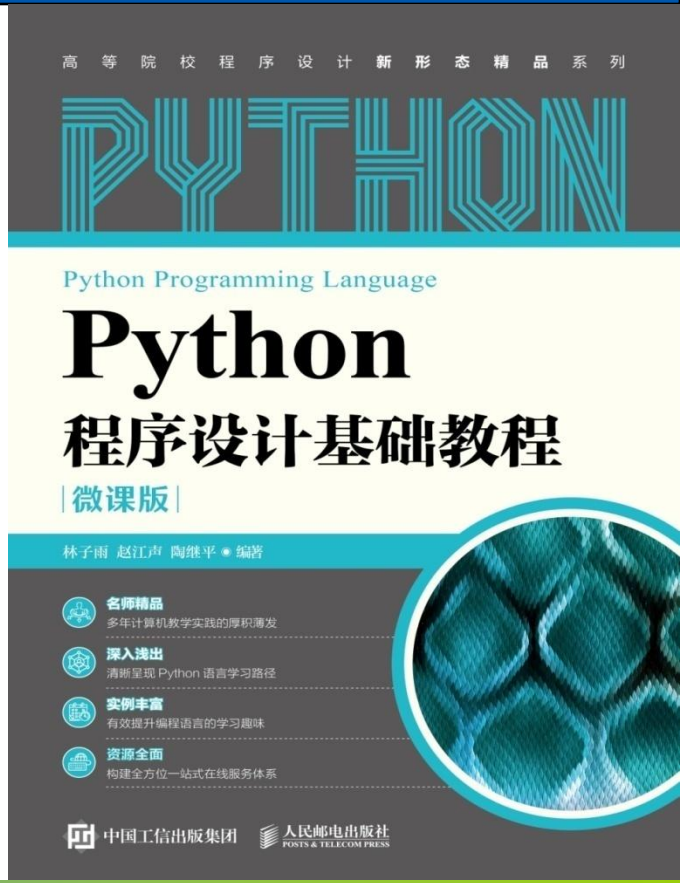
所以，包裹字符的引号的选择就取决于字符串里到底使用了哪种引号。例如，在录入一段英文的文章时，由于英文经常需要使用单引号表示缩写，比如“I'm”或者“You're”等，则包裹用的引号一般是双引号。而在平时写代码的时候，由于输入双引号需要多按一个上档键（Shift），所以建议使用单引号定义字符串。

在三引号的长字符串中，如果需要表达另一个三重引号，那么就只需将其中任意一个引号转义就行，具体实例如下：

```
>>> print('''\
'\''\
''')
'''
```

5.5 原始字符串和格式化字符串

5.5.1 原始字符串
5.5.2 格式化字符串



本PPT是如下教材的配套讲义：
《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《 Python程序设计基础教程（微课版） 》教材官方网站：
http://dblab.xmu.edu.cn/post/python

5.5.1 原始字符串

对于某些应用场景，比如输入一个文件的路径，其中会有大量的需要转义的字符“\”。如果每个字符都要输入“\\”就太麻烦了。所以，在Python里，引入了一种“原始字符串”的概念。所谓“原始字符串”，就是指在字符串前加入先导符“r”（也可以是大写的“R”），之后，字符串里的所有内容都不会被转义。具体实例如下：

```
>>> aString = r"c:\desktop\python\homework.py"
>>> print(aString)
c:\desktop\python\homework.py
>>> aString = "c:\\desktop\\python\\homework.py"
>>> print(aString)
c:\desktop\python\homework.py
```

5.5.1 原始字符串

上面实例中，首先使用了原始字符串方式，然后又使用了转义方式，可以看出，前者更加简单易懂。但这里就有一个问题，由于“\”已经没有转义的作用了，所以，在这个字符串中如果需要使用包裹字符串的引号，该如何处理呢？这时，要分为三种情况处理：

（1）如果字面量只用到一种引号，显然只需要包裹时用另一种引号就可以；
（2）如果字面量同时用两种引号，那么可以在原始字符串中使用三连引号；
（3）如果字面量里同时出现两种三连引号，就只能放弃使用原始字符串，而改为使用普通的需要转义的字符串。

5.5.1 原始字符串

下面是具体实例：

```
>>> aString = r"I'm fine, thank you"
>>> aString
"I'm fine, thank you"
>>> aString = r'''I'm fine, thank you'''
>>> aString
"I'm fine, thank you"
```

5.5.2 格式化字符串

在字符串拼接时，很多时候并不能直接简单地将数量值与字符串拼接在一起。比如，一本账单中的数值，需要保留两位小数，再比如在显示时，为了美观，需要对齐某些内容。这时，就需要使用到一种称为“格式化字符串”的特殊字符串。具体实例如下：

```
>>> aString = "{0:>4}：{1:.2f}".format("价格", 10)
>>> aString
'  价格：10.00'
```

在上面这个实例中，格式化字符串指的就是调用format函数的对象，也就是“{0:>4}：{1:.2f}”。在格式化字符串中，可以看到有一些被花括号“{}”包裹的部分，它们在Python中被称为“格式规格迷你语言”。

5.5.2 格式化字符串

这种迷你语言的基本规则如下：

{参数编号:格式化规则}

参数编号如果写为0，则对应着format函数的第一个参数，如果写1，则对应着第2个参数。每个参数可以使用多次。格式化规则的书写方法如下：

[对齐方式][符号显示规则][#][0][填充宽度][千分位分隔符][.<小数精度>][显示类型]

书写规则时，除非对应部分不出现，否则就必须严格按照上述顺序，绝对不能修改。比如，“{0:<#05}”绝对不能写成“{0:<5#0}”，必须严格按照上述顺序，这样“<”才代表对齐方式，#和0都是规则中的对应符号，5代表填写宽度，其它使用默认值。

5.5.2 格式化字符串

在格式化规则中，“对齐方式”指的是文本是居中、居左还是居右，空白部分使用什么字符填充。填充规则是：[填充文本](>|<|^|=)。比如，如果需要将文本填充进20格宽度，居右显示，空白处填写加号，就要写成：{0:+>20}。这里，“+”是填充文本，“>”指居右，数字20代表着填充宽度。具体实例如下：

```
>>> aString = "{0:+>20}".format("价格", 10)
>>> aString
'++++++++++++++++++价格'
```

在默认情况下，填充字符是空格符，可以不写。

5.5.2 格式化字符串

对齐方式里，“<”表示左对齐，“>”表示右对齐，“^”表示居中对齐。这三个符号在记忆时全部当成箭头，就比较好记，箭头指向方向就是对齐方向。“=”比较特殊，它只能用在数字上，表示填充时把填充文本放在正负号的右边，专门用来显示如“-000010”这样的文本。在使用这种对齐方式时，一般要书写填充文本0。具体的对比效果如下所示：

```
>>> aString = "{0:=+5}".format(10)
>>> aString
'+□□10'
```

表示把10输出，输出宽度为5，并且显示符号，不够5个字符的其他部分使用空格补充，=表示把补充的两个空格放在+的右侧，所以这里需要在+的右边补充两个空格.

5.5.2 格式化字符串

```
>>> aString = "{0:0=+5}".format(10)
>>> aString
'+0010'
```

表示把10输出，输出宽度为5，并且显示符号，不够5个字符的其他部分使用0补充，=表示把补充的两个0放在+的右侧，所以这里需要在+的右边补充两个0.

```
>>> aString = "{0:>+5}".format(10)
>>> aString
'□□+10'
```

表示把10输出，输出宽度为5，显示符号，右对齐，不够5个字符的其他部分使用空格补充，所以这里需要在+左边补充两个空格.

5.5.2 格式化字符串

•在上面实例中可以看到，格式化内容里书写了一个“+”，这个就对应着格式化规则中的“符号显示规则”。

•这条规则只适用于数字，对于字符串是不生效并且会提示错误的。

• “符号显示规则”的取值在默认情况下是“-”，即表示只有负数才显示符号，正数不显示。

• “符号显示规则”取值为“+”时，表示无论正数还是负数都显示符号。

• “符号显示规则”为“空格符”时，表示正数在符号位显示一个空格，负数显示负号。

5.5.2 格式化字符串

```
>>> "{0:< 5}{0:<-5}{0:<+5}{0:<5}".format(10)
'□10□□10□□□+10□□10□□□'
```

第一个格式字符串是{0:< 5}，输出宽度为5，左对齐，符号显示规则为空格，则表示正数时显示一个空格，所以，显示结果是“□10□□”。第二个格式字符串是{0:<-5}，输出宽度为5位，左对齐，符号显示规则为-，表示只有负数才显示符号，正数不会显示符号，所以显示结果为“10□□□”。第三个格式字符串是{0:<+5}，输出宽度为5，左对齐，“符号显示规则”取值为“+”时，表示无论正数还是负数都显示符号，所以显示结果为“+10□□”。第四个格式化字符串是{0:<5}，输出宽度为5，左对齐，默认情况下是“-”，即表示只有负数才显示符号，正数不显示，所以显示结果为“10□□□”。

```
>>> "{0:<5}{0:<-5}{0:<+5}{0:< 5}".format(-10)
'-10□□-10□□-10□□-10□□'
```

5.5.2 格式化字符串

在格式化规则中，在符号显示规则后面的“#”，表示如果是以2进制显示数字，则显示前导符“0b”，以8进制显示数字，则显示前导符“0o”，以16进制显示数字，则显示前导符“0x”。如果不是上述情况，就没有任何效果。具体实例如下：

```
>>> "{0:<#8b}{0:<#8o}{0:<#8d}{0:<#8x}{0:#8X}".format(10)
'0b1010□□0o12□□□□10□□□□□□0xa□□□□□□□□0XA'
```

•格式化字符串{0:<#8b}的输出结果是：0b1010□□，因为表示输出8位宽度，左对齐，使用二进制表示。

•格式化字符串{0:<#8o}的输出结果是：0o12□□□□，因为表示输出8位宽度，左对齐，使用8进制表示。

•格式化字符串{0:<#8d}的输出结果是：10□□□□□□，因为表示输出8位宽度，左对齐，使用10进制表示。

5.5.2 格式化字符串

```
>>> "{0:<#8b}{0:<#8o}{0:<#8d}{0:<#8x}{0:#8X}".format(10)
'0b1010□□0o12□□□□10□□□□□□0xa□□□□□□□□0XA'
```

•格式化字符串{0:<#8x}的输出结果是：0xa□□□□□，因为表示输出8位宽度，左对齐，使用十六进制表示。

•格式化字符串{0:#8X}的输出结果是□□□□□0XA，因为表示输出8位宽度，左对齐，使用十六进制表示。

5.5.2 格式化字符串

再次强调，格式化规则必须严格按照顺序书写，绝对不能把“#”写到数字的后面，否则会报错。如果这里还要同时加上符号显示规则，就必须写成“{0:<+#8b}”，绝对不能把“+”和“#”的位置对调，否则会报告一个错误，具体实例如下：

```
>>> "{0:<+#8b}{0:<#8o}{0:<#8d}{0:<+#08x}{0:0=+#8X}".format(10)
'+0b1010□0o12□□□□10□□□□□□+0xa0000+0X0000A'
```

•格式化字符串{0:<+#8b}的输出结果是‘+0b1010□’，因为表示输出8位宽度，左对齐，使用二进制表示（输出前导符0b），“符号显示规则”取值为“+”时，表示无论正数还是负数都显示符号，所以这里会显示+。最终不够8位，在尾部用一个空格补齐8位。

5.5.2 格式化字符串

```
>>> "{0:<+#8b}{0:<#8o}{0:<#8d}{0:<+#08x}{0:0=+#8X}".format(10)
'+0b1010□0o12□□□□10□□□□□□+0xa0000+0X0000A'
```

•格式化字符串{0:<#8o}的输出结果是'0o12□□□□'，因为表示输出8位宽度，左对齐，使用八进制表示（输出前导符0o），没有设置符号显示规则，则默认情况下是“-”，即表示只有负数才显示符号，正数不显示。最终不够8位，在尾部用4个空格补齐8位。

•格式化字符串{0:<#8d}的输出结果是'10□□□□□□'，因为表示输出8位宽度，左对齐，使用十进制表示，没有设置符号显示规则，则默认情况下是“-”，即表示只有负数才显示符号，正数不显示。最终不够8位，在尾部用6个空格补齐8位。

5.5.2 格式化字符串

```
>>> "{0:<+#8b}{0:<#8o}{0:<#8d}{0:<+#08x}{0:0=+#8X}".format(10)
'+0b1010□0o12□□□□10□□□□□□+0xa0000+0X0000A'
```

•格式化字符串{0:<+#08x}的输出结果是'+0xa0000'，因为表示输出8位宽度，左对齐，使用十六进制表示（输出前导符0x），“符号显示规则”取值为“+”时，表示无论正数还是负数都显示符号，所以这里会显示+。最终不够8位，在尾部用4个0补齐8位。

•格式化字符串{0:0=+#8X}的输出结果是'+0X0000A'，因为表示输出8位宽度，使用十六进制表示（输出前导符0X），“符号显示规则”取值为“+”时，表示无论正数还是负数都显示符号，所以这里会显示+。“=”比较特殊，它只能用在数字上，表示填充时把填充文本放在正负号的右边，这里填充文本是0，所以，在正号右边填充了4个0补齐8位。

5.5.2 格式化字符串

绝对不能把“+”和“#”的位置对调，否则会报告一个错误，具体如下：

```
>>> "{0:<#+8b}{0:<#8o}{0:<#8d}{0:<+#08x}{0:0=+#8X}".format(10)
Traceback (most recent call last):
  File "<pyshell#102>", line 1, in <module>
    "{0:<#+8b}{0:<#8o}{0:<#8d}{0:<+#08x}{0:0=+#8X}".format(10)
ValueError: Invalid format specifier
```

5.5.2 格式化字符串

•在格式化规则中，“#”后面的0（比如上面实例中的{0:<+#08x}），其含义等价于将填充字符修改为0。也就是说，“{0:<+#08x}”等价于“{0:0<+#8x}”。在0之后，输入的是填写宽度，其值是一个数字值，表示填充内容占多少的宽度。

•在格式化规则中，千分位分隔符只有两种取值，即“,”和“_”，实例如下：

```
>>> "{0:10,}{0:10_}".format(1000000)
' 1,000,000 1_000_000'
```

5.5.2 格式化字符串

在格式化规则中，在千分位分隔符之后的是小数精度，也就是显示的小数位数，不足的部分会用0补齐。比如常见的保留两位小数，就要使用这个写法。需要注意的是，在有小数位数时，必须指定这个参数是一个浮点数，否则会报错。具体实例如下：

```
>>> "{0:>010.2f}元整".format(1000000)
‘1000000.00元整'
```

格式化字符串{0:>010.2f}的含义是，右对齐，输出宽度是10位，小数点后面有两位，不足的部分用0补齐。

5.5.2 格式化字符串

在有小数位数时，必须指定这个参数是一个浮点数，否则会报错，实例如下：

```
>>> "{0:>010.2}元整".format(1000000)
Traceback (most recent call last):
  File "<pyshell#106>", line 1, in <module>
    "{0:>010.2}元整".format(1000000)
ValueError: Precision not allowed in integer format specifier
```

5.5.2 格式化字符串

在格式化规则中，最后一部分是“显示类型”，指的是数据如何呈现，根据输入数据的类型，可以分成三类，即字符串类型、整数类型和小数类型，表5-2、表5-3和表5-4给出了每种类型的可用形式。

表5-2 字符串类型的可用形式

类型	含义
‘s’	参数以字符串的形式显示。这是默认的内容，可以省略
不填写	同’s’

5.5.2 格式化字符串

表5-3 整数类型的可用形式

类型	含义
b	二进制数字
c	字符，输出时将其转换为对应的Unicode字符
d	十进制数
o	八进制数
x	十六进制数，9以上的数字用用小写字母表示
X	十六进制数，9以上的数字用大写字母表示
n	与d相似，不过它会用当前区域的设置来插入适当的数字分隔符
不填写	同d

5.5.2 格式化字符串

表5-4 小数类型的可用形式

类型	含义
e	科学计数法，小数点前有1位，小数点后取“小数精度”部分指定的数值。如“{0:.2e}”表示小数点前1位，小数点后2位。如果没有指定小数精度，则float型取6位，decimal型显示所有小数位数
E	科学计数法，与 'e' 相似，不同之处在于它使用大写字母 'E' 作为分隔字符
f	正常的小数表示，小数的位数取“小数精度”的数值。如果不指定，float型取6位，decimal型显示所有位数。如果没有小数则不显示小数和小数点
F	定点表示，与 'f' 相似，但会将 nan 转为 NAN 并将 inf 转为 INF
g	显示“小数精度”位有效数字。如果有效数字超出，则改为显示科学计数法
G	同g，但e、nan、inf会使用大写显示
n	同g，不过它会用当前区域的设置来插入适当的数字分隔符
%	会将数字乘以100并且按f显示，之后加一个“%”
不填写	同g，如果decimal类型的数字指定了context.capitals，则按对应规则显示g或者G

5.5.2 格式化字符串

下面是具体实例：

```
>>> "{0:n}".format(555555555)
'555555555'
>>> "{0:.3f}".format(555555555)
'555555555.000'
```

小数点后面保留3位小数

```
>>> "{0:.3g}".format(555555555)
'5.56e+08'
```

“显示类型”使用g时，显示“小数精度”位有效数字，但是这里有效数字超出，因此改为显示科学计数法显示

```
>>> "{:c}{:c}".format(20320,22909)
'你好‘
```

“显示类型”采用c时，输出时将其转换为对应的Unicode字符

5.5.2 格式化字符串

在使用格式化字符串时，比较麻烦的地方在于每次都要写一个format函数。Python中也为大家准备了更加简单的写法，也就是“格式化字符串字面量”，也可以称为“f-string”。特点是在字符串前加入前导符“f”或者“F”。在使用时，将格式规格迷你语言的第一部分换成变量名或者表达式就可以了，具体实例如下：

```
>>> price = 10
>>> amount = 5
>>> f"价格：{price * amount:.2f}"
'价格：50.00'
```

5.6 字符串的编码

•对于计算机而言，是不能直接保存字符的，里面只能保存二进制数。哪怕写成字节的形式，也只是一个0-255的数字。所以，这些数字哪个代表什么字符就是一个需要解决的问题。

•换而言之，需要建立一个字符与数字之间的对应关系。早在计算机出现之前，电报就使用“点”和“线”的组合来表示各个字母和数字，比如非常有名的“摩斯电码”。如果我们将“点”视为0，“线”视为1，那么在摩斯电码中，“01”就代表着字母“A”，“1000”就代表着字母B，依此类推。这样就组成了一个数字与字母的函数关系，这个关系就可以称为“字符集”。当然，摩斯电码并不适合计算机，比如“01”和“1”在计算机里都可以视为1，但是在摩斯电码中，一个是字母A，一个是字母T。

5.6 字符串的编码

•于是，我们需要一个更适合于计算机处理和显示的字符集。在早期的计算机中，这个字符集就是“美国信息互换标准代码”，即ASCII码。

•ASCII码分为控制字符和显示字符两部分。表5-5给出了ASCII码的可显示字符。

5.6 字符串的编码

表5-5 ASCII 码的可显示字符

二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形
0010 0000	32	20	（空格）（sp）	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C	<	0101 1100	92	5C	\	0111 1100	124	7C	|
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_				

5.6 字符串的编码

在Python中，可以使用chr函数将一个ASCII码可显示字符转换为其对应的数字，也可以使用ord函数进行反向转换。具体实例如下：

```
>>> chr(97)
'a'
>>> ord('a')
97
```

当然，如果要一次性转换非常多的字符，这个方法就无效了，因为“ord”函数一次只能转换一个字符。这时，就需要使用到“字节串字面量”。所谓“字面量”就是指内容本身，比如前文所述的“字符串”引号里面的内容，就可以称为该字符串的“字面量”。而这里我们需要的是将一个字符串转换为若干个字节，所以表述上就称为“字节串字面量”。

5.6 字符串的编码

•字节串字面量中每一个字节对应一个字符。由于一个字节只有8个二进制位，所以它只能表示0-255这256个字符。

•Python规定，字节串字面量只能保存ASCII码中存在的字符（包括控制字符和显示字符）以及十六进制码。书写时，需要在字符串前加字母“b”（大小写都可以）。

5.6 字符串的编码

具体实例如下：

```
>>> string = b"bytes literal"
>>> type(string)
<class 'bytes'>
>>> string[0]
98
```

从上面实例可以看出，字节串字面量的数据类型是bytes。对其进行索引，每一位返回的其实是字符的ASCII码，而不是字符。当然，也可以直接使用ASCII码书写字符串中的每一个字符。具体实例如下：

```
>>> string = "\x61\x62\x63"
>>> string
'abc’
```

5.6 字符串的编码

•在学习了以上知识之后，就可以发现一个问题。

•ASCII码平时用来显示英文是足够的，但是，对于中文、日文、韩文、阿拉伯语等其它语言而言，显然是不够用的，比如，中文的常用字就超过256个，无法用ASCII码来表示全部汉字。

•这时就需要针对特定语言设计专门的字符集。以中文为例，我国就自行研发两种适用于中文的字符集，分别是GB2312码和GBK码，它们与ASCII码的关系是，GB2312码（1980年制定）包含ASCII码，而GBK码（1995年制定）包含GB2312码。

5.6 字符串的编码

•在这里，需要特别注意一个问题，在ASCII码中，每一个字符用7位表示，而在GB2312码和GBK码中，每一个字符用16位表示。

•所以，采用GBK码或者GB2312码保存的文件，使用ASCII码方式打开时，都会显示乱码。

5.6 字符串的编码

具体实例如下：

```
>>> string = "测试字符串"
>>> gbk = string.encode("gbk")
>>> gbk
b'\xb2\xe2\xca\xd4\xd7\xd6\xb7\xfb\xb4\xae'
>>> gbk.decode("gbk")
'测试字符串'
>>> gbk.decode("ascii")
Traceback (most recent call last):
  File "<pyshell#124>", line 1, in <module>
    gbk.decode("ascii")
UnicodeDecodeError: 'ascii' codec can't decode byte 0xb2 in position 0: ordinal
not in range(128)
```

5.6 字符串的编码

•在上面实例中，使用到了两个函数encode和decode。其中，encode是编码函数，它将源字符串按照一定的编码规则转换为字节串字面量。

•ASCII码、GB2312码和GBK码的编码规则非常简单，直接把字符所对应的数字存进文件里就可以了。比如，“测”字的GBK码是“0xB2E2”，那么就直接把这个字分成两个字节存入文件。decode就是编码的反函数，作用是把字节按照编码规则反向地转换为对应字符，所以称为“解码函数”。

5.6 字符串的编码

之前曾提到，字节串字面量只能保存ASCII码中存在的字符以及十六进制码。显然，中文字符不在ASCII码中，所以显示为16进制数的形式。当然，如果两个字符集互相包含，那么就可以相互转换，具体实例如下：

```
>>> string = "测试字符串"
>>> gb2312 = string.encode("gb2312")
>>> gb2312
b'\xb2\xe2\xca\xd4\xd7\xd6\xb7\xfb\xb4\xae'
>>> gb2312.decode("gbk")
'测试字符串'
```

这也解释了在1.4.1节的第3点中，要求在行首加中文注释“# -*- coding:utf-8 -*-”的原因。这是因为在Python 2.x中，使用的默认编码规则是ASCII。因此，如果写中文的话，显然就会出现编码错误。加入这个注释，是为了让文字正确显示。那么，什么是UTF-8呢？

5.6 字符串的编码

•在说明UTF-8之前，需要先介绍一种被称为Unicode 的字符集，它是ASCII码的扩展形式。

•Unicode里包含了世界上大多数语言所需要用到的字符，从Windows NT开始，Windows系统的底层就不再是ASCII，而是Unicode。

•Unicode根据不同的编码规则，又可以分为UTF-8、UTF-16和UTF-32等多个规则。

5.6 字符串的编码

•这里需要注意Unicode与UTF-8之间的关系。

•ASCII码和Unicode都是字符集，都是为每一个字符分配一个码位，比如在ASCII码中，字符“a”的码位就是97，在Unicode中，汉字“知”的码位是30693，记为U+77E5，也可以理解成[0x77E5]。

•码位是可以直接用在输入里的，如果可以把Unicode全部记下来，就可以直接使用码位打字。比如，打开WORD文档，按住Alt键用小键盘输入30693，再放开Alt键，就可以显示出汉字“知”。

•而UTF-8、UTF-16等是针对Unicode字符集的“编码规则”，它们是将每一个“码位”转换为字节序列的规则。根据UTF-8这个规则，U+77E5应该编写为3个字节记录在文件中，分别是[0xE7]、[0x9F]和[0xA5]。

5.6 字符串的编码

与ASCII码相同，在字符串中也可以直接使用Unicode码书写字符，这时需要使用到转义字符“\uhhhh”，其中，“\u”后面接的4个十六进制字符代表两个字节。比如，“你”的Unicode编码是U+4F60，“好”的编码是U+597D。所以，“你好”就可以书写为“\u4F60”和“\u597D”，具体实例如下：

```
>>> string = "\u4f60\u597d"
>>> string
'你好'
>>> string.encode("utf-8")
b'\xe4\xbd\xa0\xe5\xa5\xbd'
```

5.6 字符串的编码

•再次强调，Unicode码与保存到文件中的、使用UTF-8转换过的二进制位是不一样的，这里一定要注意区分。在与他人沟通交流时，如果说的是编码规则，就要说UTF-8，如果说的是码表或者字符集，则要说Unicode。

•最后，选用哪种编码规则保存文件取决于个人爱好以及受众。比如制作一个面向中国的应用程序，使用UTF-8、GBK或者GB2312都是可以的。如果制作一个面向全球的应用程序，显然使用UTF-8会更合适一些，这也是在Python 3.x中默认的编码规则是UTF-8的原因。

5.7 常用操作

字符串的常用操作可以分为两种，即类型转换函数和字符串操作函数。在平时开发过程中经常用到的函数有：

（1）类型转换函数int、long、float、complex、tuple、list、chr、ord、unichr、hex和oct。根据函数名就可以判断出函数的作用，比如，int("100")就是将字符串"100"转为数字100。具体实例如下：

```
>>> int("100")
100
>>> list("I'm fine, thank you")
['I', "'", 'm', ' ', 'f', 'i', 'n', 'e', ',', ' ', 't', 'h', 'a', 'n', 'k', ' ', 'y', 'o', 'u']
```

（2）表达式转换函数eval。比如，eval("10+20+30") 就会输出数字60，具体实例如下：

```
>>> price = 50
>>> amount = 12
>>> eval("price * amount")
600
```

eval函数非常有用，需要说明的是，该函数的参数是一个表达式，也就是说可以有变量。

（3）长度计算函数len。比如，len(“123”)输出数字3。这个函数同样也是字符串的常用函数，在进行词法分析时，必定会用到。具体实例如下：

```
>>> len("price * amount")
14
```

（4）大小写转换函数lower和upper。这两个函数非常简单，作用就是把字符串里的英文字母转换为对应的大写或者小写，对非英文字母不起作用。在实际使用过程中，凡是涉及大小写不敏感的场合都需要使用本函数。比如，用户输入验证码时，大写小写都可以通过。再比如使用十六进制数时，九以上的数字使用字母表示，大小写也是一样的。具体实例如下：

```
>>> print("my python lesson".upper())
MY PYTHON LESSON
>>> print("My 1st Python Lesson".lower())
my 1st python lesson
```

（5）查询函数find。在字符串的使用过程中，查询函数可以说是最常用的函数了。比如在商品搜索过程中，用户输入了关键词，那就需要在数据库里搜索每个商品名是否包含这个关键词。查询函数find的使用方法如下：

```
>>> "It's python".find("yt")
6
>>> "It's python".find("C")
-1
>>> "It's python".find("yt", 2, 5)
-1
```

该函数的第一个参数是待查询的字符串。比如，要查询“Python课程”中是否包含“课程”二字，那么第一个参数就要写成“课程”。第二个参数是查询的开始位置，不写就是0，第三个参数是查询的结束位置，不写就是字符串的总长。查询的结果，如果有包含，则返回第一个字符在字符串中的位置，如果不包含，则返回-1。

（6）字符串分解函数split。该函数也是在开发过程中极其常用的函数。在实际使用过程中，如果用户输入了一串被精心设计过的字符串，那么往往都需要使用本函数进行分解。比如，邮箱地址的格式是“用户名@域名”，就可以使用split函数将其分解。具体实例如下：

```
>>> print("email_address@domain".split("@"))
['email_address', 'domain‘]
```

5.7 常用操作

在使用split函数的过程中，需要注意一下它的参数。该函数有两个参数，第一个参数表示分隔符，如上面的实例中的分隔符就是“@”，当然，在实际使用过程中分隔符也可能是其它字符。第二个参数是切割次数。比如，字符串“Python#C#C++#JAVA#C#”是由5门语言组成的，也就是Python、C、C++、Java和C#。但是，如果按照“#”来分隔，就会出现如下所示的结果：

```
>>> print("Python#C#C++#JAVA#C#".split("#"))
['Python', 'C', 'C++', 'JAVA', 'C', '']
```

可以看到，系统将最后的C#的“#”也视为分隔符。

在正常使用过程中，如果预计到待分隔字符串里包含分隔符时，应当要更换分隔符。不过，在本例里，也可以使用split的第二个参数解决这个问题，具体实例如下：

```
>>> print("Python#C#C++#JAVA#C#".split("#", 4))
['Python', 'C', 'C++', 'JAVA', 'C#']
```

这里的第二个参数的实际含义是处理多少个分隔符，设计成4就是处理前4个，也就是字符串会被分成5段。设置为n就说明处理前n个分隔符，字符串会被分隔为n+1段。

附录A：主讲教师林子雨简介

主讲教师：林子雨

单位：厦门大学计算机科学与技术系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://dblab.xmu.edu.cn/post/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），全国高校知名大数据教师，现为厦门大学计算机科学系副教授，厦门大学信息学院实验教学中心主任，曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国计算机学会数据库专业委员会委员，中国计算机学会信息系统专业委员会委员。国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度、2017年度和2020年度厦门大学教学类奖教金获得者，荣获2019年福建省精品在线开放课程、2018年厦门大学高等教育成果特等奖、2018年福建省高等教育教学成果二等奖、2018年国家精品在线开放课程。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学青年基金项目(No.61303004)、1项福建省自然科学青年基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题和1项2016年教育部产学协作育人项目，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。中国高校首个“数字教师”提出者和建设者，2009年至今，“数字教师”大平台累计向网络免费发布超过1000万字高价值的研究和教学资料，累计网络访问量超过1000万次。打造了中国高校大数据教学知名品牌，编著出版了中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》，并成为京东、当当网等网店畅销书籍；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过400万次，累计访问量超过1500万次。

附录B：大数据学习路线图

大数据学习路线图访问地址：http://dblab.xmu.edu.cn/post/10164/

附录C：林子雨大数据系列教材

林子雨大数据系列教材

用于导论课、专业课、实训课、公共课

了解全部教材信息：http://dblab.xmu.edu.cn/post/bigdatabook/

附录D：《大数据导论（通识课版）》教材

开设全校公共选修课的优质教材

本课程旨在实现以下几个培养目标：

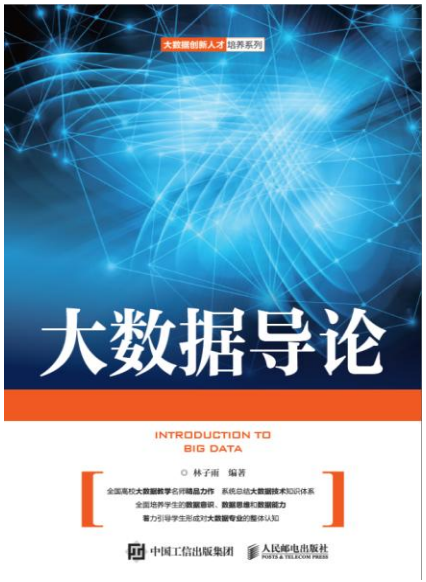
- 引导学生步入大数据时代，积极投身大数据的变革浪潮之中
- 了解大数据概念，培养大数据思维，养成数据安全意识
- 认识大数据伦理，努力使自己的行为符合大数据伦理规范要求
- 熟悉大数据应用，探寻大数据与自己专业的应用结合点
- 激发学生基于大数据的创新创业热情

高等教育出版社 ISBN:978-7-04-053577-8 定价：32元 版次：2020年2月第1版
教材官网：http://dblab.xmu.edu.cn/post/bigdataintroduction/

附录E：《大数据导论》教材

- 林子雨 编著《大数据导论》
- 人民邮电出版社，2020年9月第1版
- ISBN:978-7-115-54446-9 定价：49.80元

教材官网：http://dblab.xmu.edu.cn/post/bigdata-introduction/



开设大数据专业导论课的优质教材

扫一扫访问教材官网

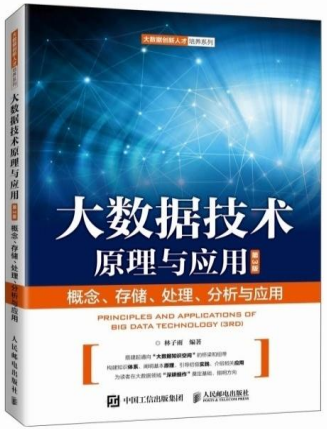
附录F：《大数据技术原理与应用（第3版）》教材

《大数据技术原理与应用——概念、存储、处理、分析与应用（第3版）》，由厦门大学计算机科学系林子雨博士编著，是国内高校第一本系统介绍大数据知识的专业教材。人民邮电出版社 ISBN:978-7-115-54405-6 定价：59.80元

全书共有17章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据 库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、Spark、流计算、Flink、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase、MapReduce、Spark和Flink等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。
欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata3

扫一扫访问教材官网



附录G：《大数据基础编程、实验和案例教程（第2版）》

本书是与《大数据技术原理与应用（第3版）》教材配套的唯一指定实验指导书

大数据教材

1+1黄金组合

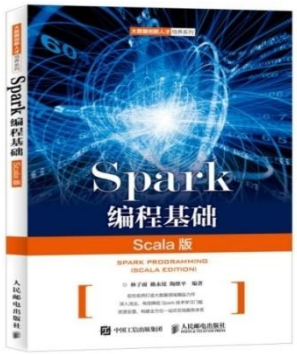
厦门大学林子雨编著

配套实验指导书

- 步步引导，循序渐进，详尽的安装指南为顺利搭建大数据实验环境铺平道路
- 深入浅出，去粗取精，丰富的代码实例帮助快速掌握大数据基础编程方法
- 精心设计，巧妙融合，八套大数据实验题目促进理论与编程知识的消化和吸收
- 结合理论，联系实际，大数据课程综合实验案例精彩呈现大数据分析全流程

林子雨编著《大数据基础编程、实验和案例教程（第2版）》
清华大学出版社 ISBN:978-7-302-55977-1 定价：69元 2020年10月第2版

附录H：《Spark编程基础（Scala版）》



《Spark编程基础（Scala版）》

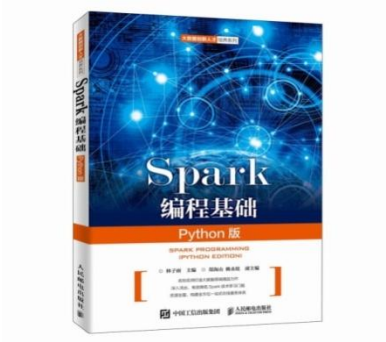
厦门大学 林子雨，赖永炫，陶继平 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9
教材官网：http://dblab.xmu.edu.cn/post/spark/

本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。

附录I：《Spark编程基础（Python版）》



《Spark编程基础（Python版）》

厦门大学 林子雨，郑海山，赖永炫 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-52439-3
教材官网： http://dblab.xmu.edu.cn/post/spark-python/

本书以Python作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Structured Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、上机实验指南等。

附录J：高校大数据课程公共服务平台

高校大数据课程

公 共 服 务 平 台

http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

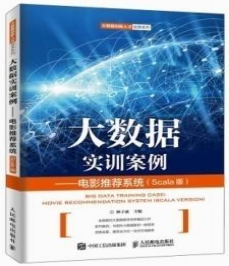
扫一扫访问平台主页

扫一扫观看3分钟FLASH动画宣传片

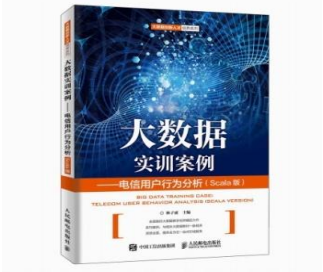
附录K：高校大数据实训课程系列案例教材

为了更好满足高校开设大数据实训课程的教材需求，厦门大学数据库实验室林子雨老师团队联合企业共同开发了《高校大数据实训课程系列案例》，目前已经完成开发的系列案例包括：

《电影推荐系统》（已经于2019年5月出版）
《电信用户行为分析》（已经于2019年5月出版）
《实时日志流处理分析》
《微博用户情感分析》
《互联网广告预测分析》
《网站日志处理分析》



系列案例教材将于2019年陆续出版发行，教材相关信息，敬请关注网页后续更新！
http://dblab.xmu.edu.cn/post/shixunkecheng/



扫一扫访问大数据实训课程系列案例教材主页


Thank You!