第六章：序列：字符串、列表和元组

目录

[前言： 1](#_Toc350501459)

[6.1 序列 2](#_Toc350501460)

[1、操作符 2](#_Toc350501461)

[2、内建函数 3](#_Toc350501462)

[6.2字符串 5](#_Toc350501463)

[1、字符串的相关操作 5](#_Toc350501464)

[2、操作符 6](#_Toc350501465)

[3、字符串与内建函数 8](#_Toc350501466)

[4、字符串的独特性 9](#_Toc350501467)

[6.3列表 10](#_Toc350501468)

[1、列表的操作 10](#_Toc350501469)

[2、列表和内建函数： 11](#_Toc350501470)

[3、列表的特殊性：使用列表构建堆栈、队列 12](#_Toc350501471)

[6.4元组 12](#_Toc350501472)

[1、元组的操作 12](#_Toc350501473)

[2、元组类型操作符和内建函数,内建方法 13](#_Toc350501474)

[3、元组的特殊性 13](#_Toc350501475)

[6.5拷贝 Python 对象 14](#_Toc350501476)

[6．6额外讲解： 15](#_Toc350501477)

[1、for else方式的循环 15](#_Toc350501478)

[2、关于数据类型不可变的通俗解释 16](#_Toc350501479)

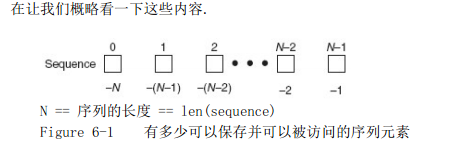
## 前言：

本章的讲解顺序：首先是讲解适用于所有序列类型的操作符和内建函数(BIFs)，然后是讲解字符串的操作符和内建函数以及他的特性和相关模块、列表的操作符和内建函数以及他的特性和相关模块、元组的操作符和内建函数以及他的特性和相关模块。

接下来我们要研究这样一些 Python 的类型，它们的成员有序排列的，并且可以通过下标

偏移量访问到它的一个或者几个成员，这类 Python 类型统称为序列，包括下面这些：字符串(普通字符串和 unicode 字符串)，列表，和元组类型。

因为这些类型其实都是由一些成员共同组成的一个序列整体，所以我们把它们统称为序列，

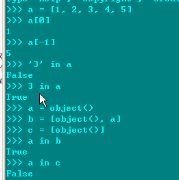


## 6.1 序列

### 1、操作符

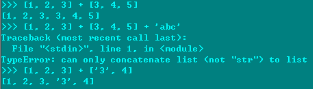
对所有序列类型都适用的操作符(优先级从高到低, 不适用于复合类型的对象比较)

* 1. 成员关系操作符: int, not in



* 1. 连接操作符: +

连接操作符左右的类型必须相同。



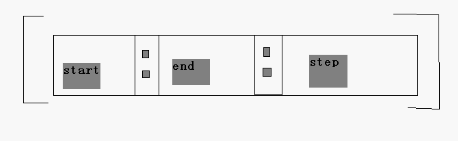
* 1. 重复操作符: \*… sequence \* int

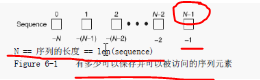
例如：



* 1. 切片操作符: (利用子序列方式结合三种操作方式, 可以非常灵活的控制序列)

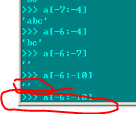
一个完整的切片一定是两个冒号，两个冒号分割出三个空间。





* + 1. [], 索引取值
    2. [start : end], 索引范围取值
    3. [::step], 切片的步长
    4. 切片操作符不会带来索引超出下标的异常

不会报错，



1.4.5例子

>>> s = 'abcdefgh'

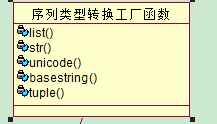
>>> s[::-1] # 可以视作"翻转"操作

'hgfedcba'

>> s[::2] # 隔一个取一个的操作

'aceg'

### 2、内建函数



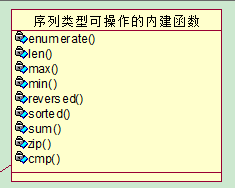
list(iter) 把可迭代对象转换为列表

str(obj) 把obj对象转换成字符串(对象的字符串表示法)

unicode(obj) 把对象转换成Unicode字符串(使用默认编码), 使用u”汉字”可以得到其unicode编码

basestring() 抽象工厂函数, 不能被实例化, 不能被调用, 仅作为str和unicode的父类

tuple(iter) 把一个可迭代对象转换成元组对象



len(),reversed()和 sum()函数只能接受序列类型对象作为参数，而剩下的则还可以接受可迭代对象做为参数，另外，max()和 min()函数也可以接受一个参数列表.

表 6.3 序列类型可用的内建函数

函数名 功能

enumerate(iterable) 生成由iterable每个元素的index值和item值组成的元组, 可以使用for key, value in enumerate的方式进行迭代

len(seq) 返回 seq 的长度

max(iterable, key=None) | max(arg0, arg1, …, key=None) 返回iterable或arg0…中的最大值, 如果要指定key, 必须是一个可以传递给sort()方法的回调函数…..要使用key, 必须使用key=method\_name的方式传参, key指定的函数接收的参数是迭代的当前元素, 在该函数中, 对元素进行处理, 返回一个对象, python会根据返回对象比较大小, 得到该结果最大的元素对应的list中的元素

min同上面的max方法. 对于max和min方法, 如果是自定义类型, 又没有指定key, 那么默认是按照id(object)的结果计算的

reversed(sequence) 返回逆序访问的迭代器

sort(cmp=None, key=None, reverse=False) 将序列进行排序, cmp指定一个接受两个参数的回调函数, 该函数得到的两个参数是序列中的两个元素, 比较将按照cmp指定的回调函数进行, 返回数字类型的比较结果, , , key指定一个接受一个参数的回调函数句柄, 该参数就是迭代到的元素, 在比较之前, 将会根据这个回调函数对要比较的元素进行一次处理, 实际参与比较的是经过该回调函数处理之后的返回值. reverse指示是否对比较结果进行逆序

sum(sequence, init=0) 返回sequence和可选参数init的总和, 等同于reduce(operator.add, sequence, init)

zip([it0, it1,... itN]) 返回一个列表，其第一个元素是 it0,it1,...这些元素的第一个元素组成的一个元组，第二个...类推.

sorted(iterable, func=None, key=None, reverse=False) 接受一个可迭代对象, 返回一个有序列表, 可选参数func, key, reverse含义和list.sort相同

zip([item0, item1, …, itemn]) 返回一个列表, [(item0, ), (item1, ), …, (itemn, )]

## 6.2字符串

### 1、字符串的相关操作

1）字符串的创建和赋值

创建一个字符串就像使用一个标量一样简单，当然你也可以把 str()作为工厂方法来创建一个字符串并把它赋值给一个变量:

>>> aString = 'Hel lo World!' # 使用单引号

>>> anotherString = "Python is cool!" # 使用双引号

>>> print aString # print 不带引号的 Hello World!

>>> anotherStrin g # 不是进行 print 操作，带有引号

'Python is cool!'

>>> s = str(ra nge(4)) # 把一个列表转换成一个字符串

>>> s

'[0, 1, 2, 3]'

2）如何访问字符串的值(字符和子串)

Python 里面没有字符这个类型,而是用长度为 1 的字符串来表示这个概念,当然，这其实也是一个子串。用方括号加一个或者多于一个索引的方式来获得子串:

3）如何改变字符串

你可以通过给一个变量赋值（或者重赋值）的方式“更新”一个已有的字符串.新的值可能与原有值差不多，也可能跟原有串完全不同。

>>> aString = aString[:6] + 'Python!'

>>> aString

'Hello Python!'

>>> aString = 'different string altogether'

>>> aString

'different string altogether'

跟数字类型一样，字符串类型也是不可变的，所以你要改变一个字符串就必须通过创建一

个新串的方式来实现。也就是说你不能只改变一个字符串的一个字符或者一个子串，然而，通过拼凑一个旧串的各个部分来得到一个新串是被允许的，正如上面你看到的那样.

4）如何删除字符和字符串

再重复一遍，字符串是不可变的,所以你不能仅仅删除一个字符串里的某个字符,你能做的

是清空一个空字符串，或者是把剔除了不需要的部分后的字符串组合起来形成一个新串。假设你想要从"Hello World!"里面删除小写的'l'

>>> aString = 'Hello World!'

>>> aString = aString[:3] + aString[4:]

>>> aString

'Helo World!'

通过赋一个空字符串或者使用 del 语句来清空或者删除一个字符串:

>>> aString = ''

>>> aString

''

>>> del aString

在大部分应用程序里，没有必要显式的删除字符串。定义这个字符串的代码最终会结束，那时 Python 会自动释放这些字符串.

### 2、操作符

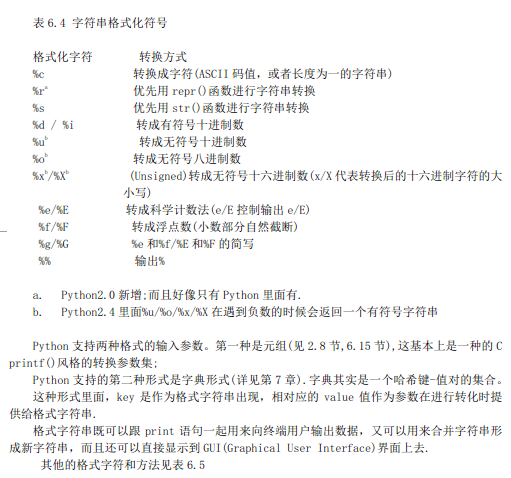
字符串的其他的操作符和序列的几乎一样:

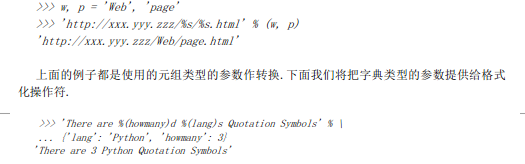
比较: 普通字符串按照ascii值比较, Unicode字符串按照unicode值比较.

在此不讲，只讲字符串特殊的操作符。

只适用于字符串的操作符

1）格式化操作符( % )





2）原始字符串操作符( r/R )

关于原始字符串的目的，在 Python1.5 里面已经有说明，是为了对付那些在字符串中出现

的特殊字符(下面的小节会介绍这些特殊字符)。在原始字符串里，所有的字符都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符。

除了原始字符串符号(引号前面的字母"r")以外,原始字符串跟普通字符串有着几乎完全相

同的语法.

这个'r'可以是小写也可以是大写，唯一的要求是必须紧靠在第一个引号前.

在三个例子的第一个例子里面，我们需要一个反斜杠加一个'n'来而不是一个换行符.：

>>> '\n'

'\n'

>>> print '\n'

>>> r'\n'

'\\n'

>>> print r'\n'

\n

3）Unicode 字符串操作符( u/U )

Unocide 字符串操作符，大写的(U)和小写的(u)是在 Python1.6 中 和 Unicode 字符串一起被引入的. 它用来把标准字符串或者是包含 Unicode 字符的字符串转换成完全地 Unicode 字符串对象。

4）字符串模板 s = Template(‘There are ${howmany} ${lang} Quotation Symbols’)

s.substitute([key = value, key = value…]) 这个函数必须提供所有的占位参数, 否则报错

s.safe\_substitute([key = value, key = value…]) 和substitute功能是一样的, 都是转成得到一个字符串, 但是这个方法对参数没有要求, 如果没有提供对应的参数, 就直接输出.

### 3、字符串与内建函数

1）序列类型公用的内建函数在字符串

cmp()：同比较操作符一样，内建的 cmp()函数也根据字符串的 ASCII 码值进行比较.

len()：内建函数 len()返回字符串的字符数.

max() and min()：返回最大或者最小的字符，(按照 ASCII 码值排列)

enumerate()：

>>> s = 'foobar'

>>> for i, t in enumerate(s):

... print i, t

...

0 f

1 o

2 o

3 b

4 a

5 r

zip()：

>>> s, t = 'foa', 'obr'

>>> zip(s, t)

[('f', 'o'), ('o', 'b'), ('a', 'r')]

2）字符串类型可操作的内建函数

raw\_input()

str() and unicode()

chr(), unichr(), and ord()

chr()函数用一个范围在 range(256)内的(就 是0到255)整数做参数,返回一个对应的字

符.unichr()跟它一样，只不过返回的是 Unicode 字符，ord()函数是 chr()函数(对于 8 位的 ASCII 字符串)或 unichr()函数(对于 Unicode 对象)的配对函数,它以一个字符(长度为 1 的字符串)作为参数,返回对应的 ASCII 数值，或者 Unicode数值。

### 4、字符串的独特性

1）三引号

然你可以用单引号或者双引号来定义字符串，但是如果你需要包含诸如换行符这样的特殊字符时，单引号或者双引号就不是那么方便了。Python 的三引号就是为了解决这个问题的，

它允许一个字符串跨多行，字符串中可以包含换行符、制表符以及其他特殊字符.三引号让程序员从引号和特殊字符串的泥潭里面解脱出来，自始至终保持一小块字符串的格式是所谓的 WYSIWYG(所见即所得)格式的。

一个典型的用例是，当你需要一块 HTML 或者 SQL 时，这时用字符串组合，特殊字符串转义将会非常的繁琐.

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)

''')

2）字符串不变性

3）Unicode p197

你写过一个例子，上面讲解的很详细。

4）Python 字符串不是通过 NUL 或者'\0'来结束的

C 编程的一个主要问题是你访问了一个字符串后面的本不属于你的空间,这种情况发生在你

没有在字符串末尾添加终结符,NUL 或者'\0'(ASCII 值为 0)的时候.Python 不仅为你自动管理内存,而且也把 C 的这个负担或者说是小麻烦去掉了.Python 中的字符串不是以 NUL 结束的,所以你不需要为是否已经添加终结符担心.字符串中只包含你所定义的东西,没有别的.

## 6.3列表

### 1、列表的操作

列表不仅可以包含 Python 的标准类型,而且可以用用户定义的对象作为自己的元素.列表可以包含不同类型的对象,而且要比 C 或者 Python 自己的数组类型(包含在 array 扩展包中)都要灵活.因为数组类型所有的元素只能是一种类型.列表可以执行 pop,empt,sort,reverse 等操作.列表也可以添加或者减少元素.还可以跟其他的列表结合或者把 一个列表分成几个.可以对单独一个元素或者多个元素执行 insert,update,或者 remove 操作.

1）如何创建列表类型数据并给它赋值

创建一个列表就像给一个变量赋值一样的简单.你手工写一个列表(空的或者有值的都行)

然后赋给一个变量,列表是由方括号([])来定义的,当然,你也可以用工厂方法来创建它.

>>> list('foo')

['f', 'o', 'o']

2）如何访问列表中的值

列表的切片操作就像字符串中一样;切片操作符([])和索引值或索引值范围一起使用

3）如何更新列表

你可以通过在等号的左边指定一个索引或者索引范围的方式来更新一个或几个元素,你也

可以用 append()方法来追加元素到列表中去.

4）如何删除列表中的元素或者列表(本身)

要删除列表中的元素,如果你确切的知道要删除元素的素引可以用 del 语句,否则可以用

remove()方法.

>>> aList

[123, 'abc', 'float replacer', ['inner', 'list'], (7-9j)]

>>> del aList[1]

>>> aList

[123, 'float replacer', ['inner', 'list'], (7-9j)]

>>> aList.remove(123)

>>> aList

['float replacer', ['inner', 'list'], (7-9j)]

### 2、列表和内建函数：

1)Cmp():

对两个列表的元素进行比较.

如果比较的元素是同类型的,则比较其值,返回结果.

如果两个元素不是同一种类型,则检查它们是否是数字.

如果是数字,执行必要的数字强制类型转换,然后比较.

如果有一方的元素是数字,则另一方的元素"大"(数字是"最小的")

否则,通过类型名字的字母顺序进行比较.

如果有一个列表首先到达末尾,则另一个长一点的列表"大".

如果我们用尽了两个列表的元素而且所 有元素都是相等的,那么结果就是个平局,就

是说返回一个 0.

2)列表类型的内建函数

P218

### 3、列表的特殊性：使用列表构建堆栈、队列

## 6.4元组

### 1、元组的操作

1）如何创建一个元组并给它赋值

>>> aTuple = (123, 'abc', 4.56, ['inner', 'tuple'], 7-9j)

>>> tuple('bar')

('b', 'a', 'r')

2）如何访问元组中的值

元组的切片操作跟列表一样,用方括号作为切片操符([]),里面写上索引值或者索引范围.

3）如何更新元组

跟数字和字符串一样,元组也是不可变类型,就是说你不能更新或者改变元组的元素,在 6.2

和 6.3.2 节里面,我们是通过现有字符串的片段再构造一个新字符串的方式解决的,对元组同样

>> tup1 = (12, 34.56)

>>> tup2 = ('abc', 'xyz')

>>> tup3 = tup1 + tup2

>>> tup3

(12, 34.56, 'abc', 'xyz')

4）如何移除一个元组的元素以及元组本身

删除一个单独的元组元素是不可能的,当然,把不需要的元素丢弃后, 重新组成一个元组是

没有问题的.

要显示地删除一整个元组,只要用 del 语句减少对象引用计数.当这个引用计数达到 0 的时

候,该对象就会被析构.记住,大多数时候,我们不需要显式的用 del 删除一个对象,一出它的作

用域它就会被析构,Python 编程里面用到显式删除元组的情况非常之少.

del aTuple

### 2、元组类型操作符和内建函数,内建方法

像列表一样 元组也没有它自己专用的运算符和内建函数.上一节中描述的列表方法都跟列

表对象的可变性有关,比如说排序,替换,添加等等,因为元组是不可变的,所以这些操作对元组

来说就是多余的,这些方法没有被实现.

### 3、元组的特殊性

1）单元素元组

曾经试过创建一个只有一个元素的元组？你在列表上试过，它可以完成，但是无论你怎么

在元组上试验，你都不能得到想要的结果。

>>> ['abc']

['abc']

>>> type(['abc']) # a list

<type 'list'>

>>>

>>> ('xyz')

'xyz'

>>> type(('xyz')) # a string, not a tuple

<type 'str'>

或许你忘记了圆括号被重载了，它也被用作分组操作符。由圆括号包裹的一个单一元素首

先被作为分组操作，而不是作为元组的分界符。一个变通的方法是在第一个元素后面添一个逗号(,)来表明这是一个元组而不是在做分组操作.

>>> ('xyz',)

2）字典的关键字

不可变对象的值是不可改变的。这就意味着它们通过 hash 算法得到的值总是一个值。这是

作为字典键值的一个必备条件。在下一章节里面我们会讨论到，键值必须是可哈希的对象，元组变量符合这个标准，而列表变量就不行。

核心笔记：列表 VS 元组

一个经常会被问到的问题是，"为什么我们要区分元组和列表变量？"这个问题也可以被表

述为“我们真的需要两个相似的序列类型吗？”，一个原因是在有些情况下，使用其中的一种类型要优于使用另一种类型。

最好使用不可变类型变量的一个情况是，如果你在维护一些敏感的数据，并且需要把这些

数据传递给一个并不了解的函数(或许是一个根本不是你写的 API),作为一个只负责一个软件某一部分的工程师，如果你确信你的数据不会被调用的函数篡改，你会觉得安全了许多。

一个需要可变类型参数的例子是，如果你在管理动态数据集合时。你需要先把它们创建出

来，逐渐地或者不定期的添加它们，或者有时还要移除一些单个的元素。这是一个必须使用可变类型对象的典型例子。幸运的是，通过内建的 list()和 tuple()转换函数,你可以非常轻松

的在两者之间进行转换. list()和 tuple()函数允许你用一个列表来创建一个元组,反之亦然.如果你有一个元组变量,但你需要一个列表变量因为你要更新一下它的对象,这时 list()函数就是你最好的帮手.如果你有一个列表变量,并且想把它传递给一个函数,或许一个 API,而你又不想让任何人弄乱你的数据,这时 tuple()函数就非常有用。

## 6.5拷贝 Python 对象

浅拷贝和深拷贝

所谓浅拷贝就是只拷贝了对对象的索引，而不是重新建立了一个对象！如果你想完全的拷

贝一个对象(包括递归，如果你的对象是一个包含在容器中的容器),你需要用到深拷贝。器类型对象没有拷贝一说。

浅拷贝: 拷贝原对象中的内容, 但是新创建对象. 比如一个list的浅拷贝就是把list中元素的引用值拷贝过去… 浅拷贝实例: 完全切片操作, 利用工厂函数, 使用copy模块的copy函数

深拷贝: 拷贝源对象中的内容, 如果某个属性(或序列中的元素)是可变对象, 把该可变对象的内容也进行拷贝。

核心模块: copy

我们刚才描述的浅拷贝和深拷贝操作都可以在 copy 模块中找到.其实 copy 模块中只有两个函数可用:copy()进行浅拷贝操作,而deepcopy()进行深拷贝。

## 6．6额外讲解：

### 1、for else方式的循环

for-else方式的循环, else子句会在所有的元素都被循环完成之后执行, 如果break, 就不执行



这与java以及c完全不同。如果换做他们的话，还需要好好的设置和一个flag变量标识。

如下：

for i in myinput[1:]:#切片操作的第一个是起始，第二个是字符串长度，第三个是步长。起始从1开始，因为第0个已经检查过了

if i not in alphanum:

print *"%s 不是一个有效的字符串"* %myinput

break

else:#当for循环完毕，仍然没有满足if的东西，及不执行break，那么就会执行else中的内容。而如果执行了break中的内容，就不会再执行else了

print *"%s 是一个有效的字符串"* %myinput

### 2、关于数据类型不可变的通俗解释

一个数据类型是不可变的,简单来讲,就意味着一旦一个对象被定义了,它的值就不能再被更新,除非重新创建一个新的对象.