第六章 序列：字符串、列表、和元组

字符串

列表

元组

6.1.2 序列类型操作符

序列类型的操作符(in/not in)

成员关系操作符(in,not in)是用来判断一个元素是否属于一个序列的。
in/not in 操作符的返回值一般来讲就是True/False。

连接操作符(+)--一般字符串使用join元组使用那个extend方法

把一个序列和另一个相同类型的序列做连接

重复操作符(*)

切片操作符([],[:],[::])

sequence[index],index可以为正数[0,1,2,3,4,5,6…]，范围[0,len(sequence)-1],也可以使用负数[0,-len(sequence)-1,-len(sequence)-2,-len(sequence)-3,-len(sequence)-4…],范围[-len(sequence),-1],切片操作符的范围为[start,end);左开右闭

```
range(start,end,setp)
print range(1,11,3)
print range(-5,-1,1)
print range(-1,-5,-1)

range用法小结--:
if start < end then step > 0
if start > end then step < 0
```

6.1.3 内建函数

类型转换

list(iter),str(obj),tuple(iter)这些函数实际上是工厂函数，将对象作为参数，并将其内容（浅）拷贝到新生成的对象中。
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！

内建函数BIFs

len(),reversed(),sum(),max(),min(),enumerate(),sorted()

6.2 字符串

与数字类型一样，字符串类型也是不可改变的
删除一个字符串使用del语句或者赋一个空字符串来实现。

6.3 字符串和操作符

序列操作符切片([]和[:])

正向索引是从0开始的,结束与总长度-1.
负向索引是从-1开始，从后往前开始计数，结束于长度的负数。
对于切片操作符[:],如果开始索引或者结束索引没有被指定，则分别以字符串的第一个和最后一个索引值为默认值。

> 注意：起始/结束索引都没有指定的话会返回整个字符串.

成员操作符in/not in

判断一个字符或者一个子串(中的字符)是否出现在另一个字符串中

```
>>> 'bc' in 'abcd' True
>>> 'n' in 'abcd' False
```

连接符(+)

运行时刻字符串连接
通过连接操作符来从原有字符串获得一个新的字符串。
建议你不要用 string 模块。原因是 Python 必须为每一个参加连接操作的字符串分配新的内存，包括新产生的字符串。
取而代之，我们推荐你像下面介绍的那样使用字符串格式化操作符(%)，或者把所有的字符串放到一个列表中去，然后用一个 join()方法来把它们连接在一起。

```
>>> '%s %s' %('Spanish', 'Inquisition')
'Spanish Inquisition'
s = ' '.join(('Spanish', 'Inquisition', 'Made Easy'))
>>> s
'Spanish Inquisition Made Easy'
```

编译时字符串连接

Python 的语法允许你在源码中把几个字符串连在一起写。

```
foo = 'Hello' 'world!'
```

通过这种方法，你可以把长的字符串分成几部分来写，而不用加反杠。同事还可以把注释加进来。

普通字符串转化为 Unicode 字符串

把一个普通字符串和一个 Unicode 字符串做连接处理，Python会把非Unicode字符转化为Unicode字符串

```
>>> 'Hello' + u' ' + 'World' + u '!'
u'Hello World!'
```

重复操作符(*)

创建一个包含了原有字符串的多个拷贝的新串

```
>>> 'Ni' * 3
'NiNiNi'
```

6.4 只适用于字符串的操作符

6.4.1 格式化操作符(%)

字符串格式化的例子P176

```
#十六进制输出:
>>> "%#x" % 108
'0x6c'
#浮点数和科学记数法形式输出
>>> '%f' % 1234.567890
'1234.567890'
>>> '%.2f' % 1234.567890
'1234.57'
>>> '%e' % 1234.567890
1.234568e+03
#整数和字符串输出
>>> "%+d" % 4
'+4'
>>> "we are at %d%%" % 100
'we are at 100%'
>>> 'Your host is: %s' % 'earth'
'Your host is: earth'
>>> "MM/DD/YY = %02d/%02d/%d" % (2, 15, 67)
'MM/DD/YY = 02/15/67'
```

令人称奇的调试工具

事实上,所有的 Python 对象都有一个字符串表示形式(通过repr()或者str()函数实现)

6.4.2 字符串模板:更简单的替代品

新式的字符串模板像现在 shell 风格的脚本语言里面那样使用美元符号($)。

```
s = Template('There are ${howmany} ${lang} Quotation Symbols')
print s.substitute(lang='Python', howmany=3)
#print s.substitute(lang='Python')
print s.safe_substitute(lang='Python')

There are 3 Python Quotation Symbols
There are ${howmany} Python Quotation Symbols
```

6.4.3 原始字符串操作符(r/R)

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。唯一的要求是必须紧靠在第一个引号前。

```
>>> print r'\n'
\n
>>> r'\n'
'\\n'
```

6.4.4 Unicode字符串操作(u/U)

```
>>> u'abc'
u'abc'
>>> ur'Hello \n World!'
u'Hello \\n World!'
```

Unicode 操作符也可以接受原始 Unicode 字符串。
Unicode 操作符必须出现在原始字符串操作符前面

6.5 内建函数

cmp(),len(),max(),min(),enumerate(),zip()

```
s, t = 'Python', 'Core'
print zip(s,t)

[('P', 'C'), ('y', 'o'), ('t', 'r'), ('h', 'e')]
```

raw_input()-Python 里面没有 C 风格的结束字符 NUL,你输入多少个字符，len()函数的返回值就是多少
str(),unicode(),chr()-[0,256),unchr()-USC2 的 Unicode范围[0,65536);USC4 的 Unicode范围[0,1114112);,ord()

6.6 字符串内建函数

```
string.count(str, beg=0,end=len(string))
string.encode(encoding='UTF-8',errors='strict')
string.decode(encoding='UTF-8',errors='strict')
string.find(str, beg=0,end=len(string))
string.index(str, beg=0,end=len(string))
string.join(seq)
string.partition(str)
string.replace(str1, str2,num=string.count(str1))
string.split(str="", num=string.count(str))
```

6.7 字符串独特特性

一些控制字符和不可打印字符(需要转义)
控制字符的一个作用是用做字符串里面的定界符，在数据库或者 web 应用中，大多数的可打印字符都是被允许用在数据项里面的，就是说可打印的字符不适合做定界符.
一个通常的解决方案是，使用那些不经常使用的，不可打印的 ASCII 码值来作为定界符,

6.7.3 字符串的不变性

```
#修改前后的id()的值是不同的
>>> s = 'abc'
>>>
>>> id(s)
135060856
>>>
>>> s += 'def'
>>> id(s)
135057968
```

6.8 Unicode

术语

Unicode用来在多种双字节字符的格式、 编码进行转换的,其中包括一些对这类字符串的操作管理功能。
ASCII 美国标准信息交换码
BOM 字节顺序标记(标识字节顺序的字符)–UTF-16编码时需要用到
UCS 通用字符集
UCS2 UCS 的双字节编码方式(见 UTF-16)
UCS4 UCS 的四字节编码方式
UTF Unicode 或者 UCS 的转换格式
UTF-8 8位 UTF 转换格式
UTF-16 16 位 UTF 转换格式

Unicode

ASCII 字符只能表示 95 个可打印字符.后来的软件厂商把ASCII 码扩展到了 8 位，这样一来它就可以多标识 128 个字符,可是 223 个字符对需要成千上万的字符的非欧洲语系的语言来说仍然太少。
Unicode 通过使用一个或多个字节来表示一个字符的方法突破了 ASCII的限制，Unicode 可以表示超过 90,000 个字符。

为了让 Unicode 和 ASCII 码值的字符串看起来尽可能的相像，ASCII 字符串成了 StringType，而 Unicode 字符串成了UnicodeType 类型。
string 模块已经不推荐使用。Python 里面处理 Unicode 字符串跟处理 ASCII 字符串没什么两样.

str(),chr()方法的运行方式:
如果一个 Unicode 字符串被作作为参数传给了 str()函数，它会首先被转换成 ASCII 字符串然后在交给 str()函数.如果该 Unicode 字符串中包含任何不被 ASCII 字符串支持的字符，会导致 str()函数报异常.
chr()也一样

这时可以用unicode()和unichr()来处理Unicode字符

Codecs

codec 是 COder/DECoder 的首字母组合.它定义了文本跟二进制值的转换方式,与ASCII不同的是Unicode用的是多字节，这就导致了存在多种不同的编码方式,如codec支持四种编码方式ASCII,ISO8859-1/Latin-1,UTF-8 和 UTF-16.

UTF-8编码
UTF-8 编码，它也用一个字节来编码 ASCII 字符，这让那些必须同时处理 ASCII码和 Unicode 码文本的程序员的工作变得非常轻松，因为 ASCII 字符的 UTF-8 编码跟 ASCII 编码完全相同。
UTF-16编码(大端小端的由来)
UTF-16把所有的字符都是用单独的一个 16 位字,两个字节来存储的。
正因为此,这两个字节的顺序需要定义一下，一般的UTF-16 编码文件都需要一个 BOM(Byte Order Mark)，或者你显式地定义 UTF-16-LE（小端）或者 UTF-16-BE(大端)字节序.

Encoding and Decoding

因为Unicode支持多种编码格式，所以每当我们向文件写入字符串的时候，必须定义一个编码(encoding参数)用于把对应的Unicode内容转换成你定义的格式，通过encode()函数可以解决这个问题。

所以， 每次我们写一个 Unicode 字符串到磁盘上我们都要用指定的编码器给他”编码”一下,相应地， 当我们从这个文件读取数据时,我们必须”解码”该文件,使之成为相应的 Unicode 字符串对象.

Unicode使用的规则

- 程序中出现字符串时一定要加个前缀 u.
- 不要用 str()函数，用 unicode()代替.
- 不要用过时的 string 模块 – 如果传给它的是非 ASCII 字符，它会把一切搞砸。
- 写入文件或数据库或者网络时，才调用 encode()函数;相应的读取时用decode()函数

pickle 的二进制格式支持不错.这点在你向数据库里面存东西是尤为突出，把它们作为 BLOB 字段存储而不是作为 TEXT 或者 VARCHAR
字段存储要好很多.万一有人把你的字段改成了 Unicode 类型，这可以避免 pickle 的崩溃.

假设你正在构建一个读写 Unicode 数据的 Web 应用.你必须确保以下方面对 Unicode 的支持:

- 数据库服务器(MySQL,PostgreSQL,SQL Server,等等)-只要确保每张表都用 UTF-8 编码就可以了
- 数据库适配器(MySQLdb 等等)–对于不是默认就支持 Unicode 模式,你必须在 connect()方法里面用一个特殊的关键字 use_unicode
  来确保你得到的查询结果是 Unicode 字符串
- Web 开发框架(mod_python,cgi,Zope,Plane,Django 等等)–mod_python 里面开启对 Unicode 的支持相当简单,只要在 request 对象里面把text-encoding 一项设成”utf-8”就行了

Python 的 Unicode 支持

unicode(),encode()/decode()
Unicode 类型:用 Unicode()工厂方法或直接在字符串前面加一个 u 或者 U 来创建实例
Unicode 序数:unichr()函数返回一个对应的 Unicode 字符

6.9 相关模块

string,re,struct,c/StringIO,base64,codecs,crpty,difflib,hashlib,hma,md5,rotor,sha,stringprep,textwrap,unicodedate

6.10 字符串关键点总结

原始字符串对每个特殊字符串都使用它的原意
Python 字符串不是通过 NUL 或者'\0'来结束的–字符串中只包含你所定义的东西,没有别的.

6.11 列表[]

创建列表类型并给他赋值

```
>>> aList = [123, 'abc', 4.56, ['inner', 'list'], 7-9j]
>>> print aList
[123, 'abc', 4.56, ['inner', 'list'], (7-9j)]
```

访问列表中的值[],[:]
更新列表

```
>>> aList[2] = 'float replacer'
>>> aList
[123, 'abc', 'float replacer', ['inner', 'list'], (7-9j)]
```

删除列表中的元素或者列表本身–del或者remove(),pop()方法来删除并从列表中返回一个特定对象

```
>>> 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)]
```

6.12 操作符

切片([],[:])
成员关系操作(in,not in)
连接接操作符(+)–把多个列表合并在一起,可以用 `extend()--把新列表添加到了原有的列表里面` 方法来代替连接操作符把一个列表的内容添加到另一个中去
向列表中添加新元素的操作用内建函数 `append()` 实现
重复操作符(*)

列表类型操作符和列表解析(属于列表的方法)

```
>>> [ i for i in range(8) if i % 2 == 0 ]
[0, 2, 4, 6]
```

6.13 内建函数

`cmp(),len(),max(),min(),sorted(),reversed(),enumerate(),zip(),sum()`
`list(),tuple()`–通过浅拷贝,常用于在这两种类型之间进行转换

6.14 列表类型的内建函数

append(),count(),extend(seq),list.index(obj,i=0,j=len(list)),
insert(index,obj),pop(index=-1),remove(obj),reverse(),
sort(func=None,key=None,reverse=False)–可以定制自己的排序方式

可以使用in和index()操作找出元素在列表中的索引值
正确的做法是应该先用 in 成员关系操作符(或者是 not in)检查一下,然后在用 index()找到这个元素的位置。因为index一个列表中不存在的元素会报错

sort()方法,它默认的排序算法是归并排序(或者说"timsort")的衍生算法,时间复杂度是 O(lg(n!)).可以通过源码查看它们的详情 – Objects/listobject.c,还有算法描述: Objects/listsort.txt.

6.15 列表的特殊特性

用列表来构建堆栈和队列

```
# -*- coding: utf-8 -*-
'''
Created by Administrator on 2016/2/28.
用列表结构来实现堆栈和队列
'''

# stack
#! /usr/bin/env python
stack = []
def pushit():
    stack.append(raw_input('Enter new string:').strip())

def popit():
    if len(stack) == 0:
        print 'Can\'t pop from an empty stack!'
    else:
        print 'Removed[','stack.pop()',']'
        stack.pop(len(stack) - 1)

def viewstack():
    print stack # calls str() internally

CMDs = {'u':pushit,'o':popit,'v':viewstack}

def showStackmenu():
    pr = """
    U-Push
    O-Pop
    V-View
```

```
    Q-Quit
    Enter choice:"""

    while True:
        while True:
            try:
                choice = raw_input(pr).strip()[0].lower()
            except(EOFError,KeyboardInterrupt,IndexError):
                choice = 'q'
            print '\nYou picked:[%s]' % choice

            if choice not in 'uovq':
                print 'Invalid option,try again'
            else:
                #break
                if choice == 'q':
                    break
                CMDs[choice]()

# queue
queue = []

def enQ():
    queue.append(raw_input('Enter new string:').strip())

def deQ():
    if len(queue) == 0:
        print 'Can\'t pop from an empty queue'
    else:
        print 'Removed[','queue.pop(0)',']'
```

```
        queue.pop(0) #去掉最后一个
def viewQ():
    print queue

queueCMDs = {'e':enQ,'d':deQ,'v':viewQ}

def showQueueMenu():
    pr = """
    E-Enqueue
    D-Dequeue
    V-View
    Q-Quit
    Enter choice:"""

    while True:
        while True:
            try:
                choice = raw_input(pr).strip()[0].lower()
            except(EOFError,KeyboardInterrupt,IndexError):
                choice = 'q'
            print '\nYou picked:[%s]' % choice

            if choice not in 'devq':
                print 'Invalid option,try again'
            else:
                #break
                if choice == 'q':
                    break
                queueCMDs[choice]()
```

```
if __name__ == "__main__":
    print 'ok'
    showQueueMenu()
    showStackmenu()
```

6.16 元组()

元组是一种不可变类型。正因为这个原因,元组能做一些列表不能做的事情，如用做一个字典的 key.另外当处理一组对象时,这个组默认是元组类型。

创建一个元组并赋值

```
>>> aTuple = (123, 'abc', 4.56, ['inner', 'tuple'], 7-9j)
>>> print aTuple
(123, 'abc', 4.56, ['inner', 'tuple'], (7-9j))
>>> emptiestPossibleTuple = (None,)
>>> print emptiestPossibleTuple
(None,)
>>> tuple('bar')
('b', 'a', 'r')
```

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。通过元组的片段再构造一个新元组

删除元组

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.

6.17 元组操作符和内建函数

创建,重复(*),连接操作(+),成员关系操作(in/not in),切片操作([],[:])

6.18 元组的特殊特性

6.18.1 不可变性(Immutability)给元组带来了什么影响?

一个数据类型成为不可变的到底意味着什么?
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。
切片操作符不能用作左值进行赋值。这和字符串没什么不同，切片操作只能用于只读的操作。(相当于prvalue)
如果我们操作从一个函数返回的元组，可以通过内建 list()函数把它转换成一个列表.

元组几个特定的行为让它看起来并不像声称的那么不可变.

虽然元组对象本身是不可变的，但这并不意味着元组包含的可变对象也不可变了。(即元组中包含了列表)

```
>>> t = (['xyz', 123], 23, -103.4)
>>> t
(['xyz', 123], 23, -103.4)
>>> t[0][1]
123
>>> t[0][1] = ['abc', 'def']
>>> t
(['xyz', ['abc', 'def']], 23, -103.4)
```

集合的默认类型为元组。
所有函数返回的多对象（不包括有符号封装的）都是元组类型。

```
def foo1():
:
return obj1, obj2, obj3
def foo2():
:
return [obj1, obj2, obj3]
def foo3():
:
return (obj1, obj2, obj3)
```

foo1()返回 3 个对象，默认的作为一个包含 3 个对象的元组类型
foo2()返回一个单一对象,一个包含 3 个对象的列表
foo3()返回一个跟 foo1()相同的对象.唯一不同的是这里的元组是显式定义的.
Note:

> 为了避免令人讨厌的副作用,建议总是显式的用圆括号表达式表示元组或者创建一个元组.

单元素元组

在列表上可以创建一个单元素的列表，而在元组上则不行。

```
>>> ['abc']
['abc']
>>> type(['abc']) # a list
<type 'list'>
>>>
>>> ('xyz')
'xyz'
>>> type(('xyz')) # a string, not a tuple
<type 'str'>
>>> ('xyz',)
('xyz',)
```

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。即('xyz')有可能会有两种不同的含义：1.作为分组操作符 2.元组分界符。
解决此问题的解决方案是在第一个元组后面加上一个逗号(,)来表明这是一个元组而不是分组操作。

作为字典键值的一个必备条件：通过 hash 算法得到的值总是一个值，这个性质成为可哈希的
而不可变对象的元组符合这个性质。
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。

为什么我们要区分元组和列表变量？我们真的需要两个相似的序列类型吗？
1.不可变量类型的例子：将一些你维护的敏感数据传递给一个并不了解的函数(另外的API)，作为只负责一个软件的某一部分的工程师，如果你确信你的数据不会被调用的函数修改，你会觉得安全了许多。
2.可变类型的例子：在管理动态数据集合时，需要不定期的添加，移除数据等操作。
通过内建函数list()，tuple()可以在两者之间转换。list()和 tuple()函数允许你用一个列表来创建一个元组,反之亦然

6.19 相关模块

copy 模块负责处理对象的浅拷贝和深拷贝。

模块名	描述
array	一种受限制的可变序列类型,要求所有的元素必须都是相同的类型。
copy	提供浅拷贝和深拷贝的能力
operator	包含函数调用形式的序列操作符,比如operator.concat(m,n)就相当于连接操作(m+n)。
re	Perl 风格的正则表达式查找
StringIO/cStringIO	把长字符串作为文件来操作，比如read(),seek()函数
Textwrap	用作包裹/填充文本的函数，也有一个类
types	包含Python支持的所有类型
collections	高性能容器数据类型

6.20 拷贝Python对象-浅拷贝和深拷贝

前面讲过，对象的赋值实际上是简单的对象引用。也就是说当你创建一个对象，然后把它赋给一个变量的时候，Python并没有拷贝这个对象，而是拷贝了这个对象的引用。

对一个对象进行**浅拷贝**其实是新创建了一个类型跟原对象一样,其内容是原来对象元素的引用。
换句话说,这个拷贝的对象本身是新的,但是它的内容不是，而是其他对象的引用。

浅拷贝和深拷贝的例子

```
>>> person = ['name', ['savings', 100.00]]
>>> hubby = person[:] # slice copy
>>> wifey = list(person) # fac func copy
>>> [id(x) for x in person, hubby, wifey]
[11826320, 12223552, 11850936]


>>> hubby[0] = 'joe'
>>> wifey[0] = 'jane'
>>> hubby, wifey
(['joe', ['savings', 100.0]], ['jane', ['savings', 100.0]])
>>> hubby[1][1] = 50.00
>>> hubby, wifey
(['joe', ['savings', 50.0]], ['jane', ['savings', 50.0]])
```

都是50.0的原因是我们仅仅做了一个浅拷贝。

序列类型对象的浅拷贝是默认类型拷贝，并可以进行一下操作：
(1)完全切片操作[:]
(2)利用工厂函数,比如 list(),dict()等
(3)使用 copy 模块的 copy 函数

下一个问题：当妻子的名字被赋值,为什么丈夫的名字没有受到影响?难道它们的名字现在不应该都是'jane'了吗?为什么名字没有变成一样的呢?
因为
在这两个列表的两个对象中,第一个对象是不可变的(是个字符串类型),而第二个是可变的(一个列表)。
正因此，当进行浅拷贝是字符串被显式的拷贝，并创建了一个字符串对象，而列表元素只是把它的引用复制了一下。

所以改变名字没有任何问题,但是更改他们银行账号的任何信息都会引发问题。

关于拷贝操作的警告:
(1)非容器类型(比如数字,字符串和其他”原子”类型的对象,像代码,类型和 xrange 对象等)没有被拷贝一说,浅拷贝是用完全切片操作来完成的.
(2)如果元组变量只包含原子类型对象,对它的深拷贝将不会进行.如果我们把账户信息改成元组类型,那么即便按我们的要求使用深拷贝操作也只能得到一个浅拷贝

目前为止只有列表可以进行深拷贝。

其实 copy 模块中只有两个函数可用:copy()进行浅拷贝操作,而 deepcopy()进行深拷贝操作。