```
#encoding=utf-8

#序列：字符串
#*****************************************
#*****************  字符串        ****************
#*****************************************

#************Part 1：  序列  **********************
#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。

#序列
#序列类型有着相同的访问模式：它的每一个元素可以通过指定一个偏移量的方式得到。
#下标偏移量是从 0  开始到  总元素数-1  结束  --  之所以要减一是因为我们是从 0  开始计数的。

#序列类型操作符

#按照优先级从高到底的顺序排列
#
#序列操作符          作用
#seq[ind]            获得下标为 ind  的元素
#seq[ind1:ind2]    获得下标从 ind1  到 ind2  间的元素集合
#seq * expr          序列重复 expr  次
#seq1 + seq2         连接序列 seq1  和 seq2
#obj in seq          判断 obj  元素是否包含在 seq  中
#obj not in seq    判断 obj  元素是否不包含在 seq  中

#成员关系操作符  (in, not in)
#成员关系操作符使用来判断一个元素是否属于一个序列的。比如对字符串类型来说，就是
#判断一个字符是否属于这个字符串，对和元组类型来说，就代表了一个对象是否属于该对象序
#列。in/not in  操作符的返回值一般来讲就是 True/False，满足成员关系就返回 True，否则返
#回 False。该操作符的语法如下：
#obj [not] in sequence

#连接操作符( + )
#这个操作符允许我们把一个序列和另一个相同类型的序列做连接。语法如下:
```

#sequence1 + sequence2
#该表达式的结果是一个包含 sequence1 和 sequence2 的内容的新序列.注意，这种方式看
#起来似乎实现了把两个序列内容合并的概念，但是这个操作不是最快或者说最有效的。对
字符
#串来说，这个操作不如把所有的子字符串放到一个列表或可迭代对象中，然后调用一个 join
#方法来把所有的内容连接在一起节约内存;类似地，对列表来说，我们推荐读者用列表类型
的
#extend()方法来把两个或者多个列表对象合并.当你需要简单地把两个对象的内容合并，或
者说
#不能依赖于可变对象的那些没有返回值(实际上它返回一个 None)的内建方法来完成的时候
时，
#连接操作符还是很方便的一个选择.

#重复操作符 (*)
#当你需要需要一个序列的多份拷贝时，重复操作符非常有用，它的语法如下：
#sequence * copies_int
#copies_int 必须是一个整数(不能是长整数).像连接操作符一样， 该操
#作符返回一个新的包含多份原对象拷贝的对象。

#切片操作符 ([], [:], [::])
#简单地讲，所谓序列类型就是包含一些顺序排列的对象的一个结构.你可以简单的用方括号
#加一个下标的方式访问它的每一个元素,或者通过在方括号中用冒号把开始下标和结束下
标分开的方式来访问一组连续的元素.
#
#访问某一个数据元素的语法如下：
#sequence[index]
#sequence 是序列的名字， index 是想要访问的元素对应的偏移量.偏移量可以是正值,范围
#从 0 到偏移最大值(比序列长度少一),用 len()函数（下一节会讲），可以得到序列长度，实
际
#的范围是 0 <= inde <= len(sequece)-1 .另外， 也可以使用负索引， 范围是 -1 到序列的负
#长度， -len(sequence), -len(sequence) <= index <= -1.正负索引的区别在于正索引以序列
#的开始为起点， 负索引以序列的结束为起点.

#内建函数(BIFs)
#序列本身就内含了迭代的概念，之所以会这样，是因为迭代这个概念就是从序列，迭代器，
或者其他支持迭代操作的对象中泛化得来的

#类型转换
#内建函数 list(),str()和 tuple()被用做在各种序列类型之间转换。你可以把它们理解成
#其他语言里面的类型转换，但是并没有进行任何的转换。这些转换实际上是工厂函数，
#将对象作为参数，并将其内容（浅）拷贝到新生成的对象中。

#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

#序列类型转换工厂函数
#函数 含义
#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)
#basestring() 抽象工厂函数,其作用仅仅是为 str 和 unicode 函数提供父类，所以不能被实例化，也不能被调用。
#tuple(iter) 把一个可迭代对象转换成一个元组对象

#Operational
#Python 为序列类型提供以下可操作 BIFs.注意，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.

#************Part 2： 字符串 **********
#Python 里面单引号和双引号的作用是相同的,Python 实际上有 3 类字符串.
#通常意义的字符串(str)和 Unicode 字符串(unicode)实际上都是抽象类 basestring 的子类.这个 basestring 是不能实例化的.

#字符串的创建和赋值
#str='Dave is DBA!'
#print(str)
#str1="dave is dba"
#print(str1)
#--双引号和单引号的作用是一样的

#访问字符串的值(字符和子串)
#str='What is Oracle!'
#print(str[0],str[4:6])

#改变字符串
#str1='dave'
#print(str1,id(str1))
#str1='oracle'
#print(str1,id(str1))
#-->
#dave 35562528
#oracle 35562080
#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

#删除字符和字符串
#字符串是不可变的,所以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。
#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.

#str1='what a fucking day'
#str1=str1[0:5]+str1[7:]
#print(str1)
#-->what fucking day

#这里通过拼接来删除，这里是创建了一个新的对象，手工 del 对象：
#del str1

#**********Part 3： 字符串和操作符 **********
#标准类型操作符
#str1='oracle'
#str2='mysql'
#print(str1>str2)
#-->True
#在做比较操作的时候，字符串是按照 ASCII 值的大小来比较的.

#序列操作符切片([ ] 和 [:])
#正向索引时，索引值开始于 0,结束于总长度减 1。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。
#str1='the big bang theory'
#print(str1[0]+','+str1[5:10]+','+str1[5:])
#-->t,ig ba,ig bang theory

#反向索引操作时,是从-1 开始，向字符串的开始方向计数，到字符串长度的负数为索引的结束.
#即：-1 表示最后一位，-2 表示倒数第二位
#str1='the big bang theory'
#print(str1[-1]+','+str1[-1:-6]+','+str1[-5:])
#-->y,,heory
#print(str1[-1]+','+str1[-6:-1]+','+str1[-5:])
#-->y,theor,heory
#这里注意，如果用方向索引，那么 str1 的顺序也必须是从左到右。 否则就不能输出结果。

#成员操作符(in ,not in)
#成员操作符用于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。出现

```
#则返回 True，否则返回 False.注意，成员操作符不是用来判断一个字符串是否包含另一个字符
#串的，这样的功能由 find()或者 index()（还有它们的兄弟：rfind()和 rindex())函数来完成

#print('bc' in 'abcd')
#-->True


#连接符( + )
#可以通过连接操作符来从原有字符串获得一个新的字符串.

#重复操作符( * )
#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
#print('Dave'*3)
#-->DaveDaveDave


#************Part 4：只适用于字符串的操作符     **********************
#
#input()
#内建的 input()函数使用给定字符串提示用户输入并将这个输入返回.
#str=input('plz input a string:')
#print(str)
#-->
#plz input a string:dave is good
#dave is good

#************Part 5：字符串的独特特性     **********************
#特殊字符串和控制字符
#像其他高级语言和脚本语言一样，一个反斜线加一个单一字符可以表示一个特殊字符,通常
#是一个不可打印的字符，这就是我们上面讨论的特殊字符，如果这些特殊字符是包含在一个原
#始字符串中的，那么它就失去了转义的功能.

#除了通常用的特殊字符，比如换行符(\n),tab 符(\t)之外，也可以直接用 ASCII 码值来标
#示特殊字符:\000 或者\xXX，分别对应字符的八进制和十六进制 ASCII 码值，下面分别是十进
#制，八进制和十六进制的 0,65,和 255:
#
#                ASCII     ASCII     ASCII
#Decimal         0           65          255
#Octal           \000       \101       \177
#Hexadecimal \x00         \x41      \xFF
#
```

#合法的 ASCII 码值范围是从 0 到 255(八进制的是 0177,十六进制是 0XFF).

#print('\x41')
#-->A

#反斜杠开头的转义字符
#/X 八进制 十进制 十六进制 字符 说明
#\0 000 0 0x00 NUL 空字符 Nul
#\a 007 7 0x07 BEL 响铃字符
#\b 010 8 0x08 BS 退格
#\t 011 9 0x09 HT 横向制表符
#\n 012 10 0x0A LF 换行
#\v 013 11 0x0B VT 纵向制表符
#\f 014 12 0x0C FF 换页
#\r 015 13 0x0D CR 回车
#\e 033 27 0x1B ESC 转义
#\" 042 34 0x22 " 双引号
#\' 047 39 0x27 ' 单引号
#\\ 134 92 0x5C \ 反斜杠

#\OOO 八进制值(范围是 000 到 0177)
#\xXX x 打头的十六进制值(范围是 0x00 到 0xFF)
#\ 连字符，将本行和下一行的内容连接起来.

#三引号
#虽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.

#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)
#''')

#************Part 6：Unicode     ************************
#Unicode 术语
#名词                意思
#ASCII          美国标准信息交换码
#BMP             基本多文种平面(第零平面)
#BOM             字节顺序标记(标识字节顺序的字符)
#CJK/CJKV        中文-日文-韩文(和越南语)的缩写
#Code point   类似于 ASCII 值，代表 Unicode 字符的值，范围在 range(1114112)或者说
0x000000 到 0x10FFFF.
#Octet          八位二进制数的位组
#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           八位 UTF 转换格式(无符号字节序列，  长度为一到四个字节)
#UTF-16          16 位 UTF 转换格式(无符号字节序列，通常是 16 位长[两个字节],见 UCS2)


#什么是 Unicode?
#Unicode 是计算机可以支持这个星球上多种语言的秘密武器.在 Unicode 之前，用的都是
#ASCII，ASCII 码非常简单，每个英文字符都是以七位二进制数的方式存贮在计算机内,其范
围
#是 32 到 126.当用户在文件中键入一个大写字符 A 时，计算机会把 A 的 ASCII 码值 65
#写入磁盘,然后当计算机读取该文件时，它会首先把 65 转化成字符 A 然后显示到屏幕上.
#ASCII 编码的文件小巧易读。一个程序只需简单地把文件的每个字节读出来，把对应的数
#值转换成字符显示出来就可以了.但是 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. 在这样机
#制下, Unicode 可以表示超过 90,000 个字符.

#在 Python 2.x 里，Python 把硬编码的字符串叫做字面上的字符串,默认所有字面上的字符串
都用 ASCII 编码，可以通过在字符串前面加一
#个'u'前缀的方式声明 Unicode 字符串，这个'u'前缀告诉 Python 后面的字符串要编码成
Unicode 字符串 .
```

#>>> "Hello World" # ASCII string
#>>> u"Hello World" # Unicode string

#Codecs 是什么?
#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,跟 ASCII 那
#种用一个字节把字符转换成数字的方式不同,Unicode 用的是多字节.这导致了 Unicode 支持多
#种不同的编码方式. 比如说 codec 支持的四种耳熟能详的编码方式是 :ASCII,ISO8859-1/Latin-1,UTF-8 和 UTF-16.
#中最著名的是 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-8 编码可以用 1 个到 4 个字节来表示其他语言的字符,CJK/East 这样的东亚文字一般都
#是用 3 个字节来表示，那些少用的、特殊的、或者历史遗留的字符用 4 个字节来表示.这给那些
#需要直接处理 Unicode 数据的程序员带来了麻烦，因为他们没有办法按照固定长度逐一读出各
#个字符.幸运的是我们不需要掌握直接读写 Unicode 数据的方法，Python 已经替我们完成了相
#关细节，我们无须为处理多字节字符的复杂问题而担心.Python 里面的其他编码不是很常用，
#事实上，我们认为大部分的 Python 程序员根本就用不着去处理其他的编码，UTF-16 可能是个例外.

#UTF-16 可能是以后大行其道的一种编码格式，它容易读写，因为它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.
#从技术上讲，UTF-16 也是一种变长编码,但它不是很常用(人们一般不会知道或者根本不
#在意除了基本多文种平面 BMP 之外到底使用的是那种平面),尽管如此，UTF-16 并不向后兼容
#ASCII,因此，实现它的程序很少，因为大家需要对 ASCII 进行支持

#编码解码
#Unicode 支持多种编码格式，这为程序员带来了额外的负担,每当你向一个文件写入字符
#串的时候，你必须定义一个编码(encoding 参数)用于把对应的 Unicode 内容转换成你定义的格
#式,Python 通过 Unicode 字符串的 encode()函数解决了这个问题，该函数接受字符串中的字符

#为参数，输出你指定的编码格式的内容。
#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.

#Python 标准库里面的绝大部分模块都是兼容 Unicode 的.除了 pickle 模块！pickle 模块只
#支持 ASCII 字符串。如果你把一个 Unicode 字符串交给 pickle 模块来 unpickle,它会报异常.
#你必须先把你的字符串转换成 ASCII 字符串才可以.所以最好是避免基于文本的 pickle 操作.
#幸运地是现在二进制格式已经作为 pickle 的默认格式了，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.

#核心模块： re
#正则表达式(RE)提供了高级的字符串模式匹配方案.通过描述这些模式的语法，你可以像使
#用“过滤器”一样高效地查找传进来的文本。这些过滤器允许你基于自定义的模式字符串抽取
#匹配模式、执行查找-替换或分割字符串.

#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.