



Python 语言程序设计

第六章

字符串的操作



字符串的格式化

01

02 字符串基本操作

字符串函数与方法

03

04 中文分词jieba

正则表达式

05



01 字符串的格式化

字符串的格式化

▲ 转义字符串与原始字符串

转义字符串 (Escape String) :

在符号前加 “\” 就构成了转义字符。

转义字符被解释器解释为另外一种含义，而不是该字符本来的含义。

```
>>> s='This is \na test!'
```

```
>>> print(s)
```

```
This is
```

```
a test!
```

```
>>>
```

转义字符

▲ 转义字符的作用

将普通字符转为特殊用途

即用来表示字符集中某些不可打印的字符，如：回车、换行等，便于输出格式的控制。

将特殊意义的字符转换为本来的含义

用来将某些标识性的特定符号，如：单引号、双引号等，说明为普通字符。

```
>>> 'Let\'s go!'
```

```
"Let's go!"
```

```
>>>
```

转义字符	功能
<code>\n</code>	在行尾的续行符，即一行未完，转到下一行继续写
<code>\\</code>	反斜线
<code>\'</code>	单引号
<code>\"</code>	双引号
<code>\a</code>	响铃
<code>\b</code>	退格(Backspace)，删除前一个字符
<code>\f</code>	分页
<code>\n</code>	换行符
<code>\r</code>	回车符
<code>\t</code>	水平制表符，用于横向跳到下一制表位
<code>\v</code>	纵向制表符
<code>\ooo</code>	3 位八进制数 <u>ooo</u> 代表的字符，如 <code>\012</code> 代表换行
<code>\xhh</code>	2 位十六进制数 <u>hh</u> 代表的字符，如 <code>\x0a</code> 代表换行
<code>\uhhhh</code>	4 位十六进制数 <u>hhhh</u> 代表的 Unicode 字符

```
>>> s='Let's do it'          #直接使用报错
SyntaxError: invalid syntax
>>> s='Let\'s do it'         #使用转义字符\
>>> print(s)
Let's do it
>>>
```

在Python中单引号(或双引号)作为字符串的标识，如果字符串中包含引号(例如 `Let's do it'`)，为了避免解释器将字符串中的引号误认为是字符串标识符，就需要对单引号进行转义。单引号 `'` 的转义字符 `\'`，尽管它由 2 个字符组成，但通常将它看做是一个整体，是一个转义字符。

【例6.1】换行符和制表符控制输出

#example6.1

s1='商品销售清单\n' ##\n为换行符

s2='商品名\t\t单价\t\t数量\t\t总价\n' #\t为水平制表符

s3='C语言程序设计\t99\t\t2\t\t198'

print(s1,s2,s3)

程序运行结果如下：

>>>

===== RESTART: C:/Users/Administrator/Desktop/example6.1.py

=====

商品销售清单

商品名	单价	数量	总价
C语言程序设计	99	2	198

>>>

原始字符

- ▲ 在字符串前加上字母 “r” 或 “R”
- ▲ 表示该字符串为原始字符串
- ▲ 原始字符串中的所有字符都具有原始含义
- ▲ 表示文件路径、URL和正则表达式中使用原始字符串

```
>>> path='C:\Windows\notepad.exe'
```

#字符串中的\n被转义为换行符

```
>>> print(path)
```

```
C:\Windows
```

```
otepad.exe
```

```
>>> path=r'C:\Windows\notepad.exe'
```

#字符串前加r，表示字符串为原字符

```
>>> print(path)
```

```
C:\Windows\notepad.exe
```

```
>>>
```

字符串的格式化%

在print()函数中，用格式字符串实现输出特定样式：

- ▲ %：格式说明符
- ▲ 格式字符串：格式说明符+普通字符
 <格式字符串>% (<值1>, <值2>, ..., <值n>)


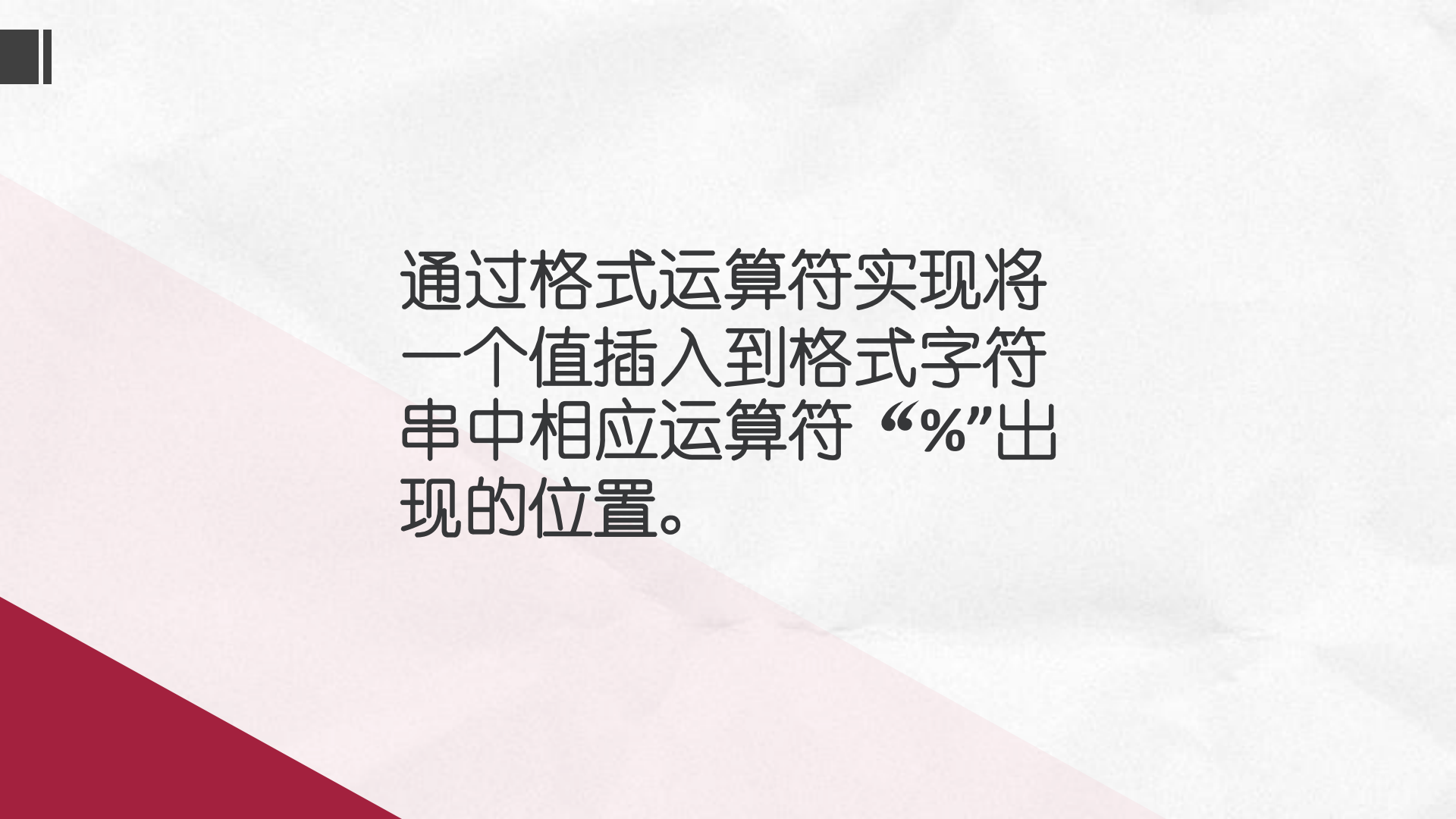
```
>>> print("今天是%d年%d月%d日，天气%s！"%(2021,7,16,'晴'))  
今天是2021年7月16日，天气晴！  
>>>
```

- 说白了：是在字符串中嵌入值，%就是占位符
- %d 啥意思？ %s 啥意思？ d代表整数，s代表字符串，详见下一页表格。

字符串的格式化%

▲ 常用格式说明符

符号	描述
%c	字符及其ASCII码
%s	字符串
%d	十进制整数
%o	八进制整数
%x	十六进制整数（用小写字母）
%X	十六进制整数（用大写字母）
%f	浮点数字，可指定小数点后的精度
%e	浮点数字，科学计数法，用小写e
%E	浮点数字，科学计数法，用大写E
%g或%G	浮点数字，根据值采用不同模式



通过格式运算符实现将
一个值插入到格式字符串
中相应运算符“%”出现的
位置。

字符串的格式化format

▲ 使用format()方法进行字符串的格式化

<模板字符串> . format (<值1>, <值2>, ..., <值n>)

0 1 2 3 0 1 2 3

```
>>> print("今天是{}年{}月{}日, 天气{}! ".format(2021,7,16,'晴'))
```

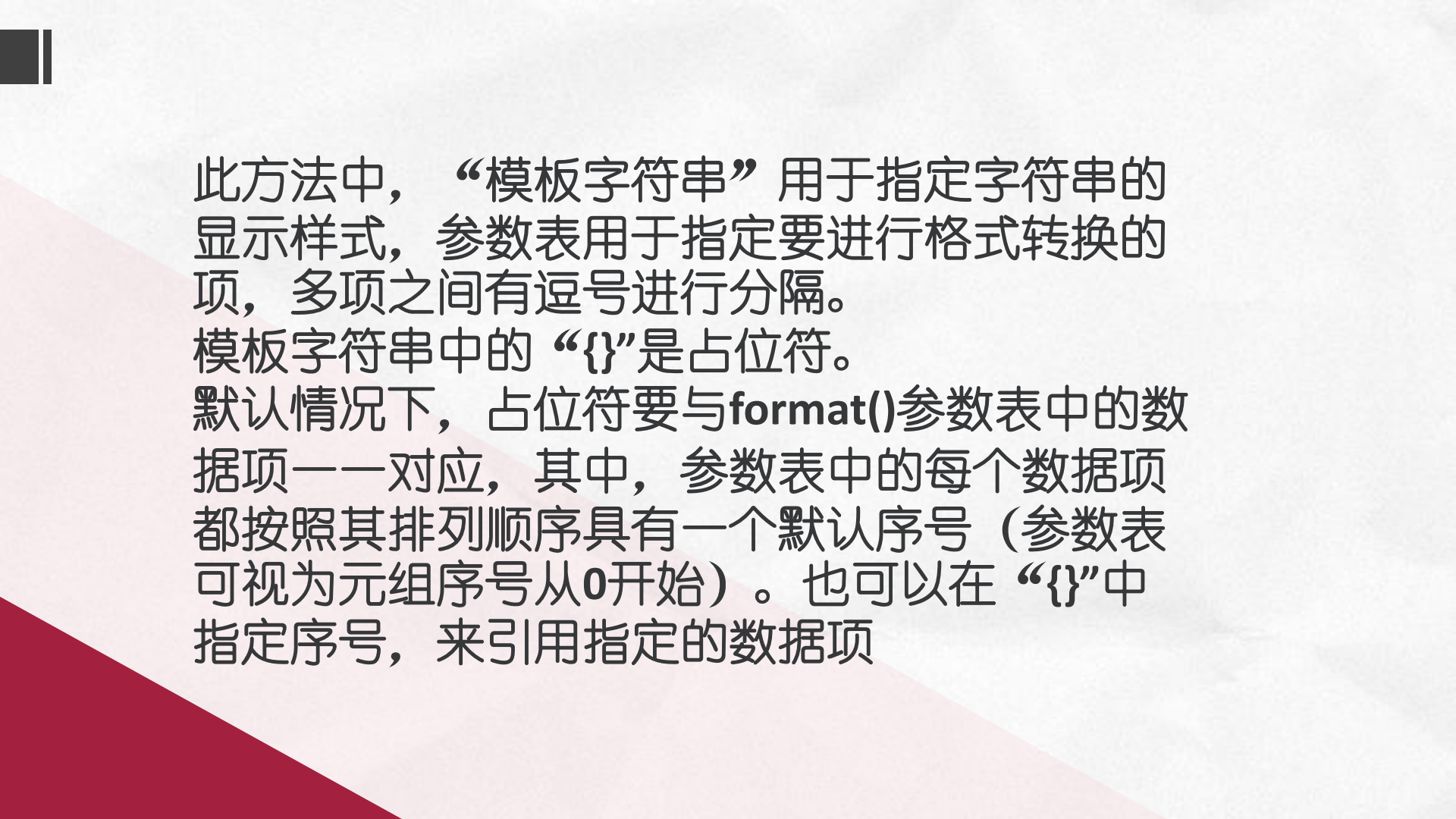
今天是2021年7月16日, 天气晴!

>>> 1 2 0 3 0 1 2 3

```
>>> print("今天是{1}月{2}日{0}年, 天气{3}! ".format(2021,7,16,'晴'))
```

今天是7月16日2021年, 天气晴!

```
>>>
```



此方法中，“模板字符串”用于指定字符串的显示样式，参数表用于指定要进行格式转换的项，多项之间有逗号进行分隔。

模板字符串中的“{}”是占位符。

默认情况下，占位符要与`format()`参数表中的数据项一一对应，其中，参数表中的每个数据项都按照其排列顺序具有一个默认序号（参数表可视为元组序号从0开始）。也可以在“{}”中指定序号，来引用指定的数据项

字符串的格式化format

▲ 使用format()方法进行字符串的格式化

<模板字符串> . format (<值1>, <值2>, ..., <值n>)

0 1 2 3 0 1 2 3

```
>>> print("今天是{}年{}月{}日, 天气{}! ".format(2021,7,16,'晴'))
```

今天是2021年7月16日, 天气晴!

>>> 1 2 0 3 0 1 2 3

```
>>> print("今天是{1}月{2}日{0}年, 天气{3}! ".format(2021,7,16,'晴'))
```

今天是7月16日2021年, 天气晴!

```
>>>
```

模板字符串,完整的语法规则:

[序号]:	<填充>	<对齐>	<宽度>	<,>	<精度>	<类型>
引导符号	用于填充的 单个字符	<左对齐 >右对齐 ^居中对齐	槽的设定 输出宽度	千分位符 仅对整数 和浮点数	浮点数小数 点位数 或 字符串宽度	整数类型 B,c,d,o,x,X 浮点数类型 E,E,f,%

"半径为{0:*^10}的圆的面积为: {1:*>10.2f}".format(r,s)

```
>>> r=125.5
>>> pi=3.1415926
>>> s=pi*r**2
>>> "半径为{0:*^10}的圆的面积为: {1:*>10.2f}".format(r,s)
'半径为**125.5**的圆的面积为: **49480.87'
>>>
```

■ 小结

1.转义字符: \n

2.原始字符: r R

3.字符的格式化: %, format()

02 字符串的基本操作

字符串的基本操作

▲ 索引

字符串中的字符按位置进行了编号，也称为索引，使用时可以通过这个编号访问字符串中的特定字符。

```
>>> str="God Wants To Check The Air Quality"
>>> str[0],str[1],str[19]
('G', 'o', 'T')
```

Python同时允许根据索引，反向访问字符串，此时字符串的编号从-1开始。

```
>>> str="God Wants To Check The Air Quality"
>>> str[-1],str[-13],str[-26]
('y', 'e', 's')
>>>
```

字符串的基本操作

▲ 分片

字符串的分片是指通过索引对字符串进行切片的操作。分片操作格式：

<字符串名>[i:j:k]

这里的 i 表示起始编号，j 表示结束编号，k 表示编号增加的步长。

注意，切片的位置不包含 j 位置上的字符。

0	1	2	3	4	5
沈	阳	师	范	大	学
-6	-5	-4	-3	-2	-1

>>> str[-1:-5:-1]

>>> str[-5:-1]

字符串的基本操作

▲ 字符串的索引与分片

分片操作格式: **<字符串名>[i:j:k]**

i 表示起始编号, j 表示结束编号(不包含), k 表示步长。i、j、k均可以省略。

i 省略时, 表示从0或-1开始; j 省略时, 表示到最后一个字符; k 省略时, 表示步长为1。

```
>>> str="God Wants To Check The Air Quality"
>>> str[4:18]
'Wants To Check'
>>> str[::2]
'GdWnsT hc h i ult'
>>> str[27::]
'Quality'
```

字符串的基本操作

▲ 字符串的索引与分片

【例】利用字符串分片操作，逆序输出字符串

#e13.1

```
st=input('输入一个字符串: ')
```

```
print("原字符串: %s"%(st))
```

```
print("逆序字符: %s"%(st[-1::-1]))
```

程序的运行结果如下:

```
>>>
```

```
===== RESTART:C:/Users/Python/5-1.py =====
```

```
输入一个字符串: 12345678
```

```
原字符串: 12345678
```

```
逆序字符: 87654321
```

```
>>>
```

字符串的基本操作

【例6.2】查询月份英文缩写。

```
#example6.2
```

```
#查询英文月份
```

```
st=""一月Jan二月Feb三月Mar四月Apr五月May六月Jun  
七月Jul八月Aug九月Sep十月Oct十一Nov十二Dec""
```

```
mon=int(input('input a month:'))
```

```
n=(mon-1)*5
```

```
print('%s的英文简称为: %s'%(st[n:n+2],st[n+2:n+5]))
```

程序的运行结果如下:

```
>>>
```

```
===== RESTART:C:/Users/Python/5-2.py
```

```
=====
```

```
input a month:6
```

```
六月的英文简称为: Jun
```

```
>>>
```

字符串的基本操作

▲ 字符串的基本运算

运算符实现几个字符串的基本运算，包括字符串的连接、判断子串、字符串的比较等。

运算符	功能
s1+s2	连接字符串s1和s2
s1*n	生成由n个s1组成的字符串
s1 in s2	如果s1是s2的子串，返回True，否则返回False
>,<==	比较字符串的Ascii码，多个字符时从左向右依次比较

字符串的基本操作

字符串的运算实例：

```
>>> s1='Python 程序设计'
```

```
>>> s2='入门'
```

```
>>> n=2
```

```
>>> s1+s2
```

```
'Python 程序设计入门'
```

```
>>> s1*n
```

```
'Python 程序设计Python 程序设计'
```

```
>>> s2 in s1
```

```
False
```

```
>>> 'a'>'A'
```

#单个字符的比较Ascii码的值，a为97，A为65

```
True
```

```
>>> 'this is a test'>'this is ' #多个字符的比较，从左向右依次每个字符
```

```
True
```

```
>>>
```

■ 小结

- 1.字符串的索引：编号从0开始
- 2.字符串的切片：正序，逆序
- 3.字符串的运算：连接，子串.....

03 字符串函数与方法

字符串的函数

▲ 字符串运算函数

调用字符串方法的格式如下：

<函数名> (<参数>)

函数名	功能
len(s)	返回集合长度
chr(x)	返回整数x对应的字符
ord(s)	返回一个字符的编码
str(x)	将数字转换为字符串

The diagram illustrates a two-dimensional square lattice. A central point is connected to its four immediate neighbors by horizontal and vertical bonds. The distance between adjacent points is labeled as 'a', representing the lattice constant. The overall structure is a regular grid of points and bonds.

#example6.5 密码隐藏加密

st0='♠♥♦♣♠♥♦♣🔥🎵🎵🎵♭🏠⌚🌀♠⌚🕒★✳️\$£¤¥'

```
print("这里是密文字符集：",st0)
```

```
st1=input("输入原密码: ")
```

```
snew=""
```

```
for s in st1:
```

if $97 \leq \text{ord}(s) \leq 122$:

```
snew=snew+st0[ord(s)-97]
```

```
elif 65 <= ord(s) <= 90:
```

```
snew=snew+st0[ord(s)-65]
```

else:

```
snew=snew+s
```

```
print("加密后的密码为: {:<}".format(snew))
```

ord(s)-97: 小写字母表中序号

st0[ord(s)-97]:取相应序号的密文

这里是密文字符集：

输入原密码: python521

加密后的密码为: ♠️🕒♣️🍀521

■ 字符串的实例

#example6.5 密码隐藏加密

st0='♠♥♦♣♠♥♦♣🔥🎵🎵🎵🎵♭🔍🔍♣♣🕒🕒★✳️\$£¤¥'

```
print("这里是密文字符集：",st0)
```

```
st1=input("输入原密码：")
```

```
snew=""
```

```
for s in st1:
```

if $97 \leq \text{ord}(s) \leq 122$:

```
snew=snew+st0[ord(s)-97]
```

```
elif 65<=ord(s)<=90:
```

```
snew=snew+st0[ord(s)-65]
```

else:

```
snew=snew+s
```

```
print("加密后的密码为: {:<}".format(snew))
```

ord(s)-65: 大写字母表中序号

st0[ord(s)-65]:取相应序号的密文

这里是密文字符集：

输入原密码: python521

加密后的密码为: ♠️🕒♣️🍀521

■ 字符串的实例

#example6.5 密码隐藏加密

st0='♠♥♦♣♠♥♦♣🔥🎵🎵🎵🎵♭🔍🔍♣♣🕒🕒★✳️\$£¤¥'

```
print("这里是密文字符集：",st0)
```

```
st1=input("输入原密码: ")
```

```
snew=""
```

```
for s in st1:
```

if $97 \leq \text{ord}(s) \leq 122$:

```
snew=snew+st0[ord(s)-97]
```

```
elif 65 <= ord(s) <= 90:
```

```
snew=snew+st0[ord(s)-65]
```

else:

```
snew=snew+s
```

```
print("加密后的密码为: {:<}".format(snew))
```

snew=snew+字符
重新连接密文字符串

这里是密文字符集：

输入原密码: python521

加密后的密码为: ♠️🕒♣️🍀521

字符串的方法

▲ 字符串运算方法

调用字符串方法的格式如下：

<字符串名>.**<方法名>** (<参数>)

```
>>> s='this is a test!'
>>> s.upper()
'THIS IS A TEST!'
>>> s.capitalize ()
'This is a test!'
>>> s.title()
'This Is A Test!'
>>>
```

字符串的方法

方法名	功能
<code>s.lower()</code>	将字符串全部转换为小写字母
<code>s.upper()</code>	将字符串全部转换为大写字母
<code>s.capitalize()</code>	将字符串的首字母转换为大写
<code>s.title()</code>	将每个单词的首字符转换为大写
<code>s.replace(old,new[,count])</code>	将s中的old字符串替换为new，count为替换的次数
<code>s.split([sep,[maxsplit]])</code>	以sep为分隔符将s拆分为一个列表，默认分隔符为空格，maxsplit表示拆分的次数，默认为-1，表示无限制
<code>s.find(s1[,start,[end]])</code>	返回s1在s中出现的位置。如果没有出现返回-1
<code>s.count(s1[,start,[end]])</code>	返回s1在s中出现的次数
<code>s.isalnum()</code>	判断s是否为全字母和数字，且至少一个字符
<code>s.isalpha()</code>	判断s是否为全字母，且至少一个字符
<code>s.isupper()</code>	判断s中是否为全大写字母
<code>s.islower()</code>	判断s中是否为全小写字母
<code>s.join(seq)</code>	将字符串序列seq中元素以s为分隔符，连接为一个新字符串

字符串的方法

▲ 字符串拆分方法的实例：

```
>>> s='this is a test!'
>>> s.split()                #将字符串以空格为分隔符拆分
['this', 'is', 'a', 'test!']
>>> s.split(sep=' ',maxsplit=1) #将字符串以空格为分隔符拆分1次
['this', 'is a test!']
>>>
```

▲ 字符串替换和查找方法的实例：

```
>>> s='this is a test!'
>>> s.find('t')              #查找字母t在字符串s中第一次出现的位置
0
>>> s.count('t')             #查找字母t在字符串s中出现的次数
3
>>> s.replace('t','T',2)     #将字符串的t替换T，替换2次
'This is a Test!'
>>>
```

【例6.6】 将原文中所有 “t” 开头的单词，替换为全大写的形式，其它单词不变。

```
#example6.6
passage='Do not trouble trouble till trouble troubles you.'
print('要转换的原文: {:<}\n'.format(passage))
word0=passage.split()
word1=[]
for w in word0:
    if w[0]=='t' or w[0]=='T':
        w=w.upper()
    word1.append(w)
passnew=" ".join(word1)
print('转换后的结果: {:<}'.format(passnew))
```

程序的运行结果如下：

```
>>>
```

```
要转换的原文: Do not trouble trouble till trouble troubles you.
```

```
转换后的结果: Do not TROUBLE TROUBLE TILL TROUBLE TROUBLES you.
```

```
>>>
```

英文词频统计

【例6.7】英文词频统计。统计给定原文中的每个单词出现的次数。

```
#example6.7
```

```
passage='Do not trouble trouble till trouble troubles you.'
```

```
print("统计原文：{:<}".format(passage))
```

```
pass1=passage.lower()
```

```
#字符串预处理
```

```
wlist=pass1.split()
```

```
#字符串拆分
```

```
counts={}
```

```
#词频统计存入字典
```

```
for w in wlist:
```

```
    counts[w]=counts.get(w,0)+1
```

```
print("统计结果：\n",counts)
```

```
#显示结果
```

程序的运行结果如下：

```
>>>
```

```
统计原文： Do not trouble trouble till trouble troubles you.
```

```
统计结果：
```

```
{'do': 1, 'not': 1, 'trouble': 3, 'till': 1, 'troubles': 1, 'you.': 1}
```

```
>>>
```

英文词频统计

一般步骤：

- 1.获取字符串：通过赋值或文件读取的方式获取原字符串。
- 2.字符串预处理：将可能影响统计结果的内容进行预处理，如标点、大小写状态等。
- 3.字符串拆分：用运算split方法将字符串拆分为单词列表。
- 4.词频统计：遍历单词列表，并利用字典进行统计。
- 5.显示结果：将字典中的统计结果，整理后显示输出。

字符串的方法

方法名	功能
<code>s.lower()</code>	将字符串全部转换为小写字母
<code>s.upper()</code>	将字符串全部转换为大写字母
<code>s.capitalize()</code>	将字符串的首字母转换为大写
<code>s.title()</code>	将每个单词的首字符转换为大写
<code>s.replace(old,new[,count])</code>	将s中的old字符串替换为new，count为替换的次数
<code>s.split([sep,[maxsplit]])</code>	以sep为分隔符将s拆分为一个列表，默认分隔符为空格，maxsplit表示拆分的次数，默认为-1，表示无限制
<code>s.find(s1[,start,[end]])</code>	返回s1在s中出现的位置。如果没有出现返回-1
<code>s.count(s1[,start,[end]])</code>	返回s1在s中出现的次数
<code>s.isalnum()</code>	判断s是否为全字母和数字，且至少一个字符
<code>s.isalpha()</code>	判断s是否为全字母，且至少一个字符
<code>s.isupper()</code>	判断s中是否为全大写字母
<code>s.islower()</code>	判断s中是否为全小写字母
<code>s.join(seq)</code>	将字符串序列seq中元素以s为分隔符，连接为一个新字符串

■ 小结

1. 字符串的函数

函数名 (参数)

2. 字符串的方法

字符串.方法名 (参数)

字符串的函数

【例】电文加密程序

```
#e13.2电文加密
original=input('输入原文: ')
crypton=""
for s1 in original:
    if s1.isalpha():
        i=ord(s1)+5
        if s1.isupper():
            if i>ord('Z'):i-=26
        else:
            if i>ord('z'):i-=26
        s2=chr(i)
        crypton+=s2
    else:
        crypton+=s1
print('输出密文: %s'%(crypton))
```

程序的运行结果如下:

```
>>>
```

```
=====
```

```
RESTART:C:/Users/Python/5-3.py
```

```
=====
```

```
输入原文: Windows!
```

```
输出密文: Bnsitbx!
```

```
>>>
```

04 中文分词jieba

The background is a dark blue-grey color. It features several abstract geometric elements: a large triangle in the top right corner composed of smaller red and orange triangles with white dots at their vertices; a similar but smaller triangle in the bottom left corner; and several thin, parallel lines in blue and white. Scattered throughout the background are small white circles and blue diamonds.



中文分词 (Chinese Word Segmentatio) 是中文自然语言处理的一个非常重要的组成部分

jieba模块是Python中优秀的中文分词第三方库，它具有强大的中文分词功能。分词原理是利用一个中文词库，确定汉字之间的关联概率，概率大的组成词组，形成分词结果。除了进行分词，用户还可以通过jieba的函数方法添加自定义的词组。

jieba作为第三方库，必须下载并用pip命令安装。

jieba库概述

- jieba作为第三方模块，首先必须使用pip命令安装后才能使用，
- 具体命令如下：

```
: \> pip install jieba
```

- jieba模块支持3种分词模式：

1. **精确模式**，试图将句子最精确地切开，适合文本分析。经过切分的中文单词经过组合，可以精确地还原为之前的文本，不存在冗余单词。
2. **全模式**，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义。分词后的单词组合起来会有冗余，不再是原来的文本。
3. **搜索引擎模式**，在精确模式的基础上，对长词再次切分，适合搜索引擎对短词语的索引和搜索，也存在冗余。

jieba库的主要函数

函数名	功能
jieba.cut(s)	精确模式，返回可迭代类型
jieba.lcut(s)	精确模式，返回列表类型
jieba.cut(s,cut_all=True)	全模式，输出文本 s 中所有可能的值，返回可迭代类型
jieba.lcut(s,cut_all=True)	全模式，返回列表类型
jieba.cut_for_search(s)	搜索引擎模式，适合搜索引擎建立索引的分词结果，粒度比较细
jieba.lcut_for_search(s)	搜索引擎模式，返回一个列表类型
jieba.add_word(w)	向分词词典中增加新词 w

三种模式分词的使用方法

```
>>> s='学而不思则罔思而不学则殆'
```

```
>>> jieba.lcut(s)
```

#不存在冗余

```
['学而不思', '则', '罔', '思而', '不学则', '殆']
```

```
>>> jieba.lcut(s,cut_all=True)
```

#所有可能成词的词语，冗余较多

```
['学而不思', '则', '罔', '思', '而', '不学', '则', '殆']
```

```
>>> jieba.lcut_for_search(s)
```

#精确模式后，再次拆分，存在冗余。

```
['学而不思', '则', '罔', '思而', '不学', '不学则', '殆']
```

```
>>>
```

向词典中增加新词

```
>>> jieba.add_word("思而不学")
>>> jieba.lcut(s)
['学而不思', '则', '罔', '思而不学', '则', '殆']
>>> jieba.lcut(s, cut_all=True)
['学而不思', '则', '罔', '思而不学', '不学', '则', '殆']
>>> jieba.lcut_for_search(s)
['学而不思', '则', '罔', '不学', '思而不学', '则', '殆']
>>>
```

推荐使用lcut()函数

(1) 无论是哪种模式，cut()函数与lcut()函数分词的结果是相同的，区别是返回值的类型不同。由于lcut()函数返回的列表类型比较灵活，因此经常被使用。

(2) 精确模式的结果是完整的，而且没有多余；全模式返回的结果包含所有可能的中文词语，比较全面，但冗余也最大；搜索引擎模式首先执行精确模式，然后再对结果中的长词进一步切分，适合搜索引擎的检索。

中文分词与词频统计

中文词频统计的一般步骤可以归纳如下：

第一步，获取文本。通过字符串赋值或文件读取获取字符串。

第二步，分词处理。将文本拆分并存入相应的数据结构中，如列表等。

第三步，词频统计。遍历数据结构逐词统计出现次数，存入数据结构，如字典等。

第四步，结果再加工。对统计数据筛选或加工，删除无效数据，如副词、连词和标点等。

第五步，输出显示。按条件对统计结果进行分析，并输出显示出来。

【例6.8】统计《红楼梦》出场次数最多的前10位人物

```
#example6.8
import jieba
txt = open("红楼梦.txt", "r", encoding='utf-8').read()
words = jieba.lcut(txt)           #精确分词，返回一个列表
counts = {}                       #用字典类型存储人物及出现的次数
for word in words:
    if len(word) != 1:            #不统计单个汉字的词汇
        counts[word] = counts.get(word,0) + 1
listitem = list(counts.items())    #将字典转换为列表
#按列表中每个元组第二个元素的值（即人数）降序排序
listitem.sort(key=lambda x:x[1], reverse=True)
print("{0:<10}{1:>8}".format("人物","出场次数"))
for i in range(10):               #输出排序后的列表前10项
    word, count = listitem[i]
    print ("{0:-<10}{1:->10}".format(word, count))
```

程序运行结果如下

```
>>>
```

```
=====RESTART:C:\Python\example6.8.py=====
```

```
Building prefix dict from the default dictionary ...
```

```
Loading model from cache C:\Users\LLQ\AppData\Local\Temp\jieba.cache
```

```
Loading model cost 0.740 seconds.
```

```
Prefix dict has been built successfully.
```

人物	出场次数
----	------

宝玉-----	3687
---------	------

什么-----	1590
---------	------

一个-----	1415
---------	------

我们-----	1205
---------	------

贾母-----	1204
---------	------

那里-----	1170
---------	------

凤姐-----	1096
---------	------

王夫人-----	1007
----------	------

如今-----	991
---------	-----

你们-----	988
---------	-----

```
>>>
```

【例6.81】 【例6.8】 进行优化

对结果中的无关数据进行删除
例如，介词、代词和副词等

```
#example6.8-1
import jieba
txt = open("红楼梦.txt", "r", encoding='utf-8').read()
```

#分词并统计

```
words = jieba.lcut(txt)          #精确分词，返回一个列表
counts = {}                     #用字典类型存储人物及出现的次数
for word in words:
    if len(word) != 1:          #不统计单个汉字的词汇
        counts[word] = counts.get(word,0) + 1
```

#建立排除词库，删除与人名无关的词汇

```
excludes = {"什么","一个","我们","你们","他们","那里","这里",\
            "如今","说道","知道","众人","出来","起来","奶奶",\
            "一面","自己","只见","没有","怎么","不知","不是",\
            "这个","听见","这样","进来","两个","告诉","就是",\
            "东西","咱们","回来"}
```

```
for word in excludes:
```

```
    del(counts[word])
```

#将字典转换为列表，按人数降序排序

```
listitem = list(counts.items())
listitem.sort(key=lambda x:x[1], reverse=True)
```

#输出排序后的列表前10项

```
print("{0:<10}{1:>8}".format("人物","出场次数"))
for i in range(10):
    word, count = listitem[i]
    print (" {0:-<10}{1:->10}".format(word, count))
```

程序运行结果如下

```
>>>
```

```
=====RESTART:C:\Python\example6.8-1.py=====
```

```
Building prefix dict from the default dictionary ...
```

```
Loading model from cache C:\Users\LLQ\AppData\Local\Temp\jieba.cache
```

```
Loading model cost 0.733 seconds.
```

```
Prefix dict has been built successfully.
```

人物	出场次数
----	------

宝玉-----	3687
---------	------

贾母-----	1204
---------	------

凤姐-----	1096
---------	------

王夫人-----	1007
----------	------

老太太-----	961
----------	-----

姑娘-----	928
---------	-----

太太-----	820
---------	-----

贾璉-----	667
---------	-----

平儿-----	589
---------	-----

袭人-----	571
---------	-----

```
>>>
```

【例6.8-2】再次进行优化

```
#example6.8-2
import jieba
txt = open("红楼梦.txt", "r", encoding='utf-8').read()

#添加自定义词典
add = ["宝姐姐", "李纨", "二奶奶"]
for i in add:                                #将人物添加到jieba词典
    jieba.add_word(i)

#分词并统计
words = jieba.lcut(txt)                      #精确分词，返回一个列表
counts = {}                                  #用字典类型存储人物及出现的次数
for word in words:
    if len(word) != 1:                        #不统计单个汉字的词汇
        #几个重要人物称谓的整合
        if word in ["林妹妹", "林姑娘", "黛玉"]: word = "林黛玉"
        if word in ["宝姑娘", "宝姐姐", "宝钗"]: word = "薛宝钗"
        if word in ["二奶奶", "琏二奶奶", "凤姐", "凤姐儿"]: word = "王熙凤"
        if word in ["宝玉", "宝二爷"]: word = "贾宝玉"
        if word in ["老太太", "老祖宗"]: word = "贾母"
        if word in ["太太"]: word = "王夫人"
    counts[word] = counts.get(word, 0) + 1
```

【例6.8-2】再次进行优化

```
#建立排除词库，删除与人名无关的词汇
excludes = {"什么","一个","我们","你们","他们","那里","这里",\
            "如今","说道","知道","众人","出来","起来","奶奶",\
            "一面","自己","只见","没有","怎么","不知","不是",\
            "这个","听见","这样","进来","两个","告诉","就是",\
            "东西","咱们","回来"}

for word in excludes:
    del(counts[word])
#将字典转换为列表，按人数降序排序
listitem = list(counts.items())          #将字典转换为列表
listitem.sort(key=lambda x:x[1], reverse=True)
#输出排序后的列表前10项
print("{0:<10}{1:>8}".format("人物","出场次数"))
for i in range(50):
    word, count = listitem[i]
    print ("{0:-<10}{1:->10}".format(word, count))
```

程序运行结果


```
>>>
=====RESTART:C:\Python\example6.8-2.py=====
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\LLQ\AppData\Local\Temp\jieba.cache
Loading model cost 0.739 seconds.
Prefix dict has been built successfully.
人物           出场次数
贾宝玉-----3798
贾母-----2244
王夫人-----1827
王熙凤-----1784
林黛玉-----1027
姑娘-----928
贾琏-----667
薛宝钗-----662
平儿-----589
袭人-----571
>>>
```

■ 小结

1. jieba的3种分词模式
2. 中文词频统计的一般步骤
3. 中文分词实例

05 正则表达式





正则表达式 (Regular Expression) 描述了一种字符串匹配的模式，由普通字符（例如字符 a 到 z）以及特殊字符（称为元字符）组成的文字模式。

正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

用来检查一个字符串是否含有某种子串、将匹配的子串做替换或者从某个字符串中取出符合某个条件的子串等等。

■ 小结

在工作中我们经常遇到这样的需求：

1. 给你一个字符串, 把字符串里面的
链接、数字、电话等显示不同的颜色；
2. 给你一个包含自定义表情的文字，找出里
面的表情，替换成本地的表情图片；
3. 根据用户的输入内容，判断是否是微信号、
手机号、邮箱、纯数字等；

THANK YOU

The user can demonstrate on a projector or computer, or print the presentation and make it into a film to be used in a wider field

文本文件的操作

唐宋以来，以回文体入诗之风很盛。宋代文学家苏轼有《题织锦图回文》诗云：

春晚落花余碧草，夜凉低月半梧桐。
人随雁远边城暮，雨映疏帘绣阁空。

倒读则诗曰：
空阁绣帘疏映雨，暮城边远雁随人。
桐梧半月低凉夜，草碧余花落晚春。

回文诗是中国文化中心一个特殊形式，也是世界上为中国仅有的一种艺术形式。

```
>>>
===== RESTART: C:/Users/
=
输入一句诗文：春晚落花余碧草
回文：草碧余花落晚春
>>>
```

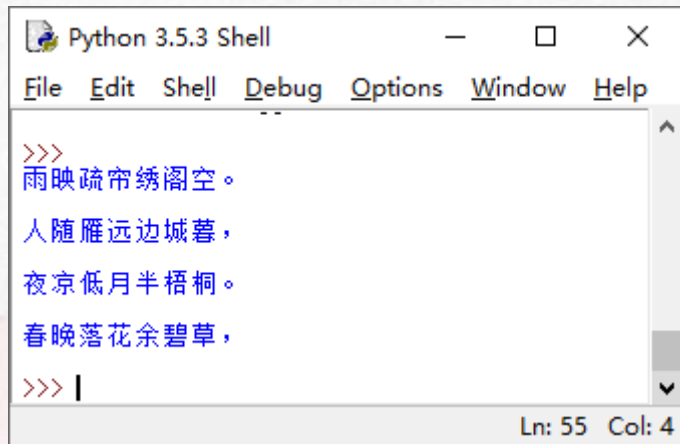
```
#单句回文
行=input("输入一句诗文：")
print("回文：{}".format(行[-1::-1]))
```

文本文件的操作

```
#0整句回文  
f=open("题织锦图.txt","r",encoding='UTF-8')  
f1=open("题织锦图回文.txt","w")  
整诗=f.readlines()
```

```
for s in 整诗[-1::-1]:  
    print(s)  
    f1.write(s)  
f.close()  
f1.close()
```

春晚落花余碧草，
夜凉低月半梧桐。
人随雁远边城暮，
雨映疏帘绣阁空



```
Python 3.5.3 Shell  
File Edit Shell Debug Options Window Help  
>>>  
雨映疏帘绣阁空。  
人随雁远边城暮，  
夜凉低月半梧桐。  
春晚落花余碧草，  
>>> |  
Ln: 55 Col: 4
```

文本文件的操作

#0整句回文

```
f=open("题织锦图.txt","r",encoding='UTF-8')
```

```
f1=open("题织锦图回文.txt","w")
```

```
整诗=f.readlines()
```

```
for s in 整诗[-1::-1]:
```

```
    print(s)
```

```
    f1.write(s)
```

```
f.close()
```

```
f1.close()
```

打开文件f，读的方式，“r”

打开文件f1，写的方式，“w”

在f中读出全文，形成列表=>整诗

遍历列表：整诗的逆序

显示输出

写入文件f1中

关闭文件f

关闭文件f1

文本文件的操作

#2整篇回文

```
f=open("题织锦图.txt","r",encoding='UTF-8')
```

```
f1=open("题织锦图回文.txt","w")
```

```
整诗=f.readlines()
```

```
for s in 整诗[-1::-1]:
```

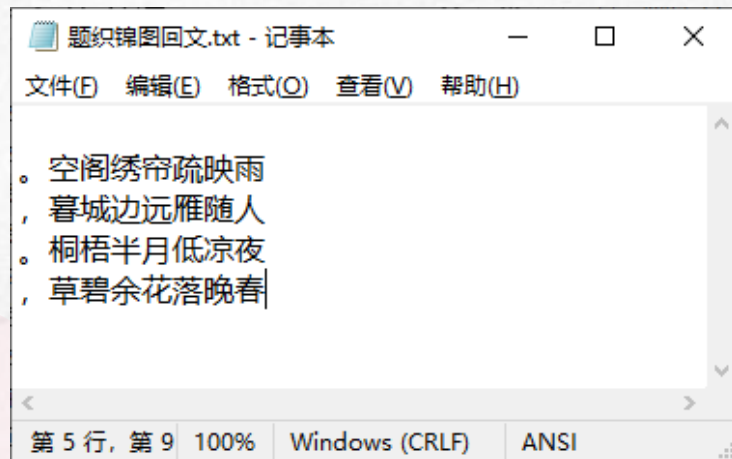
```
    print( ? ? )
```

```
    f1.write( ? ? )
```

```
f.close()
```

```
f1.close()
```

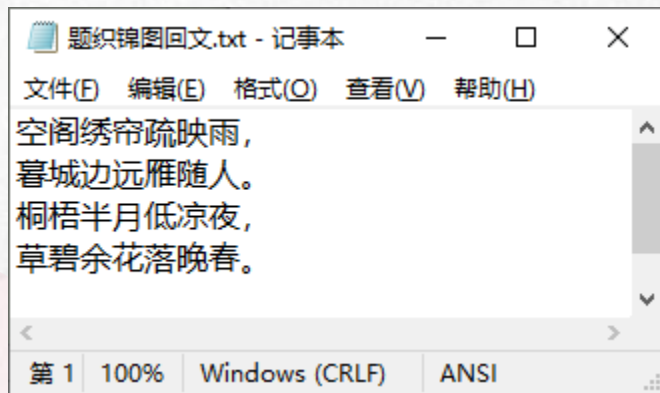
春晚落花余碧草，
夜凉低月半梧桐。
人随雁远边城暮，
雨映疏帘绣阁空



文本文件的操作

?

春晚落花余碧草，
夜凉低月半梧桐。
人随雁远边城暮，
雨映疏帘绣阁空



文件的打开：open()函数

Python通过解释器内置的open()函数打开一个文件，并实现该文件与一个程序变量的关联，open()函数格式如下：

<变量名> = open(<文件名>, <打开模式>)

打开模式	含义
'r'	只读模式，如果文件不存在，返回异常FileNotFoundError，默认值
'w'	覆盖写模式，文件不存在则创建，存在则完全覆盖源文件
'x'	创建写模式，文件不存在则创建，存在则返回异常FileExistsError
'a'	追加写模式，文件不存在则创建，存在则在原文件最后追加内容
'b'	二进制文件模式
't'	文本文件模式，默认值
'+'	与r/w/x/a一同使用，在原功能基础上增加同时读写功能

文件内容的读取

方法	含义
<code><file>.readall()</code>	读入整个文件内容，返回一个字符串或字节流*
<code><file>.read(size)</code>	从文件中读入整个文件内容，如果给出参数，读入前size长度的字符串或字节流
<code><file>.readline(size)</code>	从文件中读入一行内容，如果给出参数，读入该行前size长度的字符串或字节流
<code><file>.readlines(hint)</code>	从文件中读入所有行，以每行为元素形成一个列表，如果给出参数，读入hint行

文件的写入方法

1.write()

将字符串写入文件，在使用该函数前，打开文件函数open()不能以' r'的方式使用。
write()函数的调用格式如下：

<文件对象>.write(<变量>)

2.writelines()

将一个列表的内容都写入到文件中。函数调用格式如下：

<文件对象>.writelines (<列表>)

其中，参数列表为字符串列表，函数writelines()将字符串列表写入文件中。

文件的关闭

<文件对象名>.close()