第5章 字符串与正则表达式

学习目标

- 掌握字符串的构造方法。
- 理解字符的编码及编码与字符的相互转换方法。
- 熟练掌握字符串的格式化方法。
- 掌握字符串前特定字符的作用。
- 掌握字符串的截取方法。
- 掌握适用于字符串的常用内置函数及字符串对象的常用方法。
- 熟悉 string 模块中的函数和常量。
- 了解正则表达式的概念和元字符的用法。

在 Python 中，字符串主要有 str 类型的对象、io 模块中的 StringIO 对象和 array 模块中的 array 对象。str 类型的字符串是不可变对象，一旦创建完成，该对象就不可再修改。io 模块中的 StringIO 对象和 array 模块中的 array 对象均为可变的对象，对象中的内容可以修改。限于篇幅，本书不对这两种可变类型的字符串对象展开阐述。若没有特别说明，本书中的字符串均指 str 类型的字符串。

字符串是一类特殊的数据集对象，是指以一对引号（单引号、双引号或三引号）为边界的字符序列。引号之间的字符序列是字符串的内容。str 类型的字符串是一种不可变序列。既然字符串属于序列，那么字符串就支持序列的一系列通用操作，如元素访问、切片、成员测试、计算长度等。但因为字符串是不可变的，针对字符串的某些操作就会有所限制，如不能使用切片来修改字符串中的字符等。字符串也是一种可迭代对象。

本章首先介绍字符串对象的构造，然后介绍字符串编码方式，再介绍字符串格式化、以特定字符为前缀的字符串、字符串的截取、适用于字符串的常用函数及字符串对象的常用方法、string 模块的基本用法，最后简单介绍正则表达式的概念与用法。


视频讲解

5.1 字符串构造

在 Python 中，字符串的构造主要通过两种方法来实现。一种是使用 str 类来构造字符串对象；另一种是以成对的单引号、双引号或三引号为边界符，直接将字符序列括起来。使用引号是一种非常便捷的构造字符串的方式。

1. 用 str 类来构造字符串

可以用 str(object='')或 str(bytes_or_buffer[,encoding])两种格式来构造字符串。第

一种格式是从一个对象 object 来构造字符串。例如：

```
>>> x = str()                          #生成一个空字符串
>>> x
''
>>> len(x)
0
>>> str(10)
'10'
>>> y = str([1,2,3])
>>> y
'[1, 2, 3]'
>>> len(y)                             #列表中每个逗号后面自动添加了一个空格,所以长度为 9
9
>>>
```

第二种格式在 5.2 节介绍。

2. 单引号或双引号构造字符串

在用单引号或双引号作为边界符构造字符串时，要求引号成对出现。如 'Python World!'、'ABC'、"what is your name?"，都是构造字符串的方法。

'string"在 Python 中不是一个合法的字符串，会提示以下错误信息：

```
>>> a = 'string"
SyntaxError: EOL while scanning string literal
```

3. 单双引号构造字符串的特殊用法

如果代码中的字符串包含了单引号，且不用转义字符，那么整个字符串就要用双引号作为边界符来构造，否则就会出错。例如：

```
>>> "Let's go!"
"Let's go!"
>>> print("Let's go!")                 #用 print()函数输出更直观
Let's go!
>>> 'Let's go!'
SyntaxError: invalid syntax
```

如果代码中的字符串包含了双引号，且不用转义字符，那么整个字符串要用单引号作为边界符来构造。例如：

```
>>> '"Hello world!",he said.'
'"Hello world!",he said.'
>>> print('"Hello world!",he said.')   #用 print()函数输出更直观
"Hello world!",he said.
```

4. 字符串中引号的转义

字符串中引号的转义，可以解决如下的错误。例如：

```
>>> 'Let's go!'
SyntaxError: invalid syntax
```

如果按照如下方式来表示就是可以的：

```
>>> 'Let\'s go!'
"Let's go!"
>>> print('Let\'s go!')
Let's go!
```

上面代码中的反斜线“\”对字符串中的引号进行了转义,表示反斜线后的单引号是字符串中的一个普通字符,而不是构造字符串的边界符。如下例子为利用反斜线对其后面的双引号进行转义。

```
>>> "\"Hello world!\"he said"
'"Hello world!"he said'
>>> print("\"Hello world!\"he said")
"Hello world!"he said
```

5. 转义字符

转义字符以“\”开头,后接某些特定的字符或数字。Python 中常用的转义字符如表 5.1 所示。

表 5.1 Python 中常用的转义字符

转义字符	含义	转义字符	含义
\(位于行尾)	续行符	\v	纵向(垂直)制表符
\\	一个反斜杠(\)	\f	换页符
\'	单引号(')	\ooo	3 位八进制数 ooo 对应的字符
\"	双引号(")	\xhh	2 位十六进制数 hh 对应的字符
\n	换行符	\uhhhh	4 位十六进制数 hhhh 表示的 Unicode 字符
\r	回车		
\t	横向(水平)制表符		

示例:

```
>>> print("你好\n再见!")          #\n 表示换行,相当于按 Enter 键
你好
再见!
>>> print("你好我好\t大家都很好\t爱你们")
你好我好    大家都很好    爱你们
>>> print('\123\x6a')            #八进制数 123 对应的字符是 S,十六进制数 6a 对应的字符 j
Sj
```

6. 原始字符串

假设在 C:\test 文件夹中有一个文件夹 net,如何输出完整路径呢?可能你想到的是:

```
>>> print("C:\test\net")
c:   est
et
```

怎么会是这样的输出呢?原来字符串中“\t”和“\n”都表示转义字符。

正确的路径应如何表示呢?

第 1 种方法:使用“\\”表示反斜杠,则 t 和 n 不再形成\t 和\n。例如:

```
>>> print("C:\\test\\net")
c:\test\net
```

第 2 种方法：使用原始字符串。例如：

在字符串前面加上字母 r 或 R 表示原始字符串，所有的字符都是原始的本义而不会进行任何转义。例如：

```
>>> print(r"C:\test\net")
c:\test\net
```

7. 三重引号字符串

三重引号字符串是一种特殊的用法。三重引号将保留所有字符串的格式信息。如字符串跨越多行，行与行之间的回车符、引号、制表符或者其他任何信息，都将保存下来。在一对三重引号之间可以自由地使用单引号和双引号。例如：

```
>>> '''"What's your name?"
      "My name is Jone"'''
'"What\'s your name?"\n "My name is Jone"'
>>> print('''"What's your name?"
      "My name is Jone"''')
"What's your name?"
      "My name is Jone"
```

需要注意的是，当作为字符串内容的字符序列中最后一个字符为双引号时，如果字符串边界符使用三重引号，要使用三重单引号；如果字符串边界符使用三重双引号，则需要在作为字符串内容的并且是最后一个字符的双引号前加反斜杠，对其进行转义。反之，当字符串内容中的最后一个字符为单引号时，如果字符串边界符使用三重引号，要使用三重双引号；否则，作为字符串最后一个位置上内容的单引号前需要添加反斜杠来进行转义。例如：

```
>>> y = """Let's say:"Hello World!\""""        #字符串内容中最后一个字符为双引号
>>> y
'Let\'s say:"Hello World!"'
>>> print(y)
Let's say:"Hello World!"
```

【例 5.1】 编写程序，分别用双引号、单引号和三引号作为字符串边界符，实现语句 Let's say："Hello World!"的正确输出。

程序源代码如下：

```
#example5_1.py
#coding = utf - 8
print("Let's say:\"Hello World!\"")
print('Let\'s say:"Hello World!"')
print('''Let's say:"Hello World!"''')
```

程序 example5_1.py 的运行结果如下：

```
>>>
 ============ RESTART: G:\ example5_1.py ============
Let's say:"Hello World!"
Let's say:"Hello World!"
Let's say:"Hello World!"
```

5.2 字符串编码

任何字符在计算机中都是以特定的编码来表示、存储和传输的。为了方便相互之间的信息交换和识别,在计算机领域先后制定了多种编码方式。

ASCII(American Standard Code for Information Interchange,美国信息交换标准代码)是基于拉丁字母的一套计算机编码系统,主要用于显示现代英语和其他西欧语言。它是现今最通用的单字节编码系统,并等同于国际标准 ISO/IEC 646。

ASCII 使用指定的 7 位或 8 位二进制数组合来表示 128 或 256 种可能的字符。标准 ASCII 也叫基础 ASCII 码,使用 7 位二进制数(剩下的最高位为二进制 0)来表示所有的大写和小写字母、数字 0 ～9、标点符号及在英语中使用的特殊控制字符。128～255 为扩展 ASCII。

标准 ASCII 与字符的对照关系如表 5.2 所示。

表 5.2 标准 ASCII 与字符的对照关系

ASCII	字符	ASCII	字符	ASCII	字符	ASCII	字符	ASCII	字符	ASCII	字符
0	NUL	22	SYN	44	,	66	B	88	X	110	n
1	SOH	23	ETB	45	−	67	C	89	Y	111	o
2	STX	24	CAN	46	.	68	D	90	Z	112	p
3	ETX	25	EM	47	/	69	E	91	[	113	q
4	EOT	26	SUB	48	0	70	F	92	\	114	r
5	ENQ	27	ESC	49	1	71	G	93	]	115	s
6	ACK	28	FS	50	2	72	H	94	^	116	t
7	BEL	29	GS	51	3	73	I	95	_	117	u
8	BS	30	RS	52	4	74	J	96	`	118	v
9	HT	31	US	53	5	75	K	97	a	119	w
10	LF	32	(space)	54	6	76	L	98	b	120	x
11	VT	33	!	55	7	77	M	99	c	121	y
12	FF	34	"	56	8	78	N	100	d	122	z
13	CR	35	#	57	9	79	O	101	e	123	{
14	SO	36	$	58	:	80	P	102	f	124	|
15	SI	37	%	59	;	81	Q	103	g	125	}
16	DLE	38	&	60	<	82	R	104	h	126	～
17	DC1	39	'	61	=	83	S	105	i	127	DEL
18	DC2	40	(	62	>	84	T	106	j		
19	DC3	41	)	63	?	85	U	107	k		
20	DC4	42	*	64	@	86	V	108	l		
21	NAK	43	+	65	A	87	W	109	m		

GB2312 是我国制定的简体中文编码,GBK 是对 GB2312 的扩充。GBK 编码在 Windows 内部对应其代码页(Code Page)为 cp936。这些编码均使用 2 字节表示简体中文。

不同国家有不同的语言,也就有不同的编码。日本制定了 Shift_JIS 编码、韩国制定了

Euc-kr 编码，而不同的编码格式之间差别很大。并且同一编号在不同的编码体系中可能表示不同的字符。如果一篇文章既有英文又有中文，还有日文，那无论采用上述哪种编码，都可能会出现乱码。

Unicode 编码把所有语言的字符都统一到一套编码里，这样就不会再有乱码问题了。对于 ASCII 中的字符，Unicode 采用与 ASCII 相同的编码，只是将编码的长度由 8 位扩展为 16 位；而对于其他语言文字的字符，全部重新统一编码。最常用的 Unicode 编码是用 2 字节表示一个字符，对有些字符需要 3～4 字节的编码来表示。

采用 Unicode 编码，乱码问题是没有了，但新的问题又出来了，如果有一篇文章全是英文，用 Unicode 编码比 ASCII 编码需要多一倍的存储空间，这样既浪费存储空间又影响传输效率。为了解决 Unicode 编码的存储和传输问题，提出了 UTF(Unicode Transformation Format，Unicode 转换格式)编码。

UTF 编码规定了 Unicode 字符串的存储与传输格式。UTF 家族中使用最广泛的是 UTF-8，还有 UTF-16、UTF-32 等。UTF-8 编码是“可变长编码”，它可以使用 1～4 字节的编码表示一个符号。编码的长度根据不同的字符而有所变化。UTF-8 编码以 1 字节表示一个英语字符(兼容 ASCII)，以 3 字节表示一个中文字符，还有一些语言使用 2 字节或 4 字节表示一个字符。从 Unicode 到 UTF-8 并不是直接对应，而是要经过一些算法和规则来转换的。

Python 中 str 表示字符串类型，支持 Unicode 编码；bytes 表示字节串类型。两者均是不可变对象类型。str 类型和 bytes 类型经常需要相互转换。例如，字符串需要转换为字节串在网络上进行传输，对方接收到字节串后需要转换为字符串呈现出来。

str 类里有一个 encode()方法，根据字符串生成由参数中 encoding 指定的编码方式编码的 bytes 类型的字节串。例如：

```
>>> help(str.encode)
Help on method_descriptor:

encode(self, /, encoding = 'utf - 8', errors = 'strict')
    Encode the string using the codec registered for encoding.

    encoding
      The encoding in which to encode the string.
    errors
      The error handling scheme to use for encoding errors.
      The default is 'strict' meaning that encoding errors raise a
      UnicodeEncodeError. Other possible values are 'ignore', 'replace' and
      'xmlcharrefreplace' as well as any other name registered with
      codecs.register_error that can handle UnicodeEncodeErrors.
```

bytes 类有个 decode()方法，将字节串使用参数 encoding 指定的编码方式解码成为字符串 str 类型的数据。例如：

```
>>> help(bytes.decode)
Help on method_descriptor:

decode(self, /, encoding = 'utf - 8', errors = 'strict')
```

```
    Decode the bytes using the codec registered for encoding.

    encoding
      The encoding with which to decode the bytes.
    errors
      The error handling scheme to use for the handling of decoding errors.
      The default is 'strict' meaning that decoding errors raise a
      UnicodeDecodeError. Other possible values are 'ignore' and 'replace'
      as well as any other name registered with codecs.register_error that
      can handle UnicodeDecodeErrors.

>>> s = '我'
>>> s
'我'
>>> type(s)                          #s 是 str 类型字符串
<class 'str'>
>>> s1 = s.encode('gbk')             #编码成字节串 bytes 类型,GBK 编码格式
>>> s1
b'\xce\xd2'
>>> type(s1)                         #s1 是字节串 bytes 类型
<class 'bytes'>
>>> s2 = s.encode('utf-8')           #编码成字节串 bytes 类型,UTF-8 编码格式
>>> s2
b'\xe6\x88\x91'
>>> type(s2)                         #s2 是字节串 bytes 类型
<class 'bytes'>
>>> s3 = s1.decode('gbk')            #使用 GBK 进行解码,将字节串转换为字符串
>>> s3
'我'
>>> type(s3)                         #s3 是字符串 str 类型
<class 'str'>
>>> s4 = s2.decode('utf-8')          #使用 UTF-8 进行解码,将字节串转换为字符串
>>> s4
'我'
>>> type(s4)                         #s4 是字符串 str 类型
<class 'str'>
>>> s5 = s.encode('ascii')           #中文字符串不能以 ASCII 编码
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    s5 = s.encode('ascii')
UnicodeEncodeError: 'ascii' codec can't encode character '\u6211' in position 0: ordinal not in
range(128)
>>> 'ABC'.encode('ascii')            #英文字符串可以以 ASCII 编码
b'ABC'
```

另外,也可以使用 bytes 类的对象构造方法 bytes(string,encoding[,errors]) 和 str 类的对象构造方法 str(bytes_or_buffer[,encoding[,errors]])完成字节串和字符串的相互构造。可以用 bytes(string,encoding[,errors])根据参数中的字符串 string 来构造 bytes 类型的字节串。可以使用 str(bytes_or_buffer[,encoding[,errors]])根据参数中的字节串 bytes_or_buffer 来构造 str 类型的字符串。例如:

```
>>> s = '阳光'
>>> s
'阳光'
>>> type(s)                              # 字符串 str 类型
<class 'str'>
>>> b = bytes(s, encoding = 'gbk')
>>> b
b'\xd1\xf4\xb9\xe2'
>>> type(b)                              # 字节串 bytes 类型
<class 'bytes'>
>>> u = bytes(s, encoding = 'utf - 8')
>>> u
b'\xe9\x98\xb3\xe5\x85\x89'
>>> type(u)
<class 'bytes'>
>>> bs = str(b, encoding = 'gbk')
>>> bs
'阳光'
>>> type(bs)                             # str 类型
<class 'str'>
>>> us = str(u, encoding = 'utf - 8')
>>> us
'阳光'
>>> type(us)                             # str 类型
<class 'str'>
>>>
```

Python 3 中，字符串默认采用 Unicode 编码，支持中文字符，无论是数字字符、英文字母、汉字等都按一个字符来对待和处理。例如：

```
>>> s1 = '大熊猫'
>>> len(s1)                              # 字符个数
3
>>> s2 = 'I like 这 18 只大熊猫'
>>> len(s2)
14
>>>
```

另外，在 Python 3. x 中可以使用中文作为标识符。例如：

```
>>> 你好 = 'abc'
>>> 你好
'abc'
    >>>
```

可以使用 sys 模块中的 getdefaultencoding()函数获取 Python 环境系统(Python 解释器)对 Unicode 字符串当前使用的默认编码类型。例如：

```
>>> import sys
>>> sys.getdefaultencoding()
'utf - 8'
>>>
```

5.3 字符串格式化

使用 print()函数很容易输出各种对象,但 print()函数无法输出设计复杂的格式。用加号拼接字符串常量和字符串变量可以生成满足某些格式要求的字符串,但通常需要复杂或大量的程序代码。Python 提供了字符串格式化的方法,使得程序可以在字符串中嵌入变量并定义变量代入的格式。这样可以定义并生成复杂格式的字符串。本节介绍利用%运算符、字符串的 format()方法、f-string 字面量方法进行字符串格式化的过程。


视频讲解

5.3.1 用%格式化字符串

用%进行字符串格式化涉及两个概念:格式定义和格式化运算。字符串内部格式的定义以%开头。字符串后面的格式化运算符%表示用其后面的对象代替格式串中的格式,最终得到一个字符串。

字符串格式化的一般形式如图 5.1 所示。这里字符串中只给出了一个格式定义。字符串中,格式定义的两端均可以有普通字符或其他字符串格式的定义。

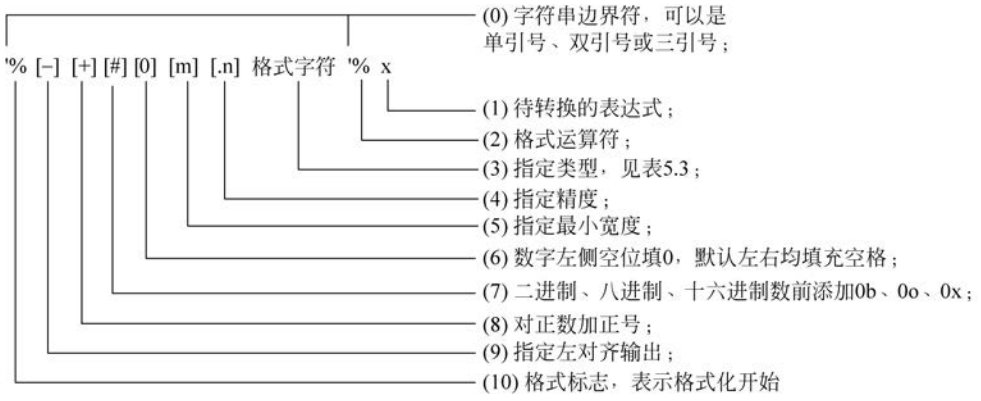

图 5.1 字符串格式化的一般形式

1. 字符串格式的书写

(1) [ ]中的内容可以省略。

(2) 简单的格式是%加格式字符,如%f、%d、%c 等。

(3) 当最小宽度及精度都出现时,它们之间不能有空格,格式字符和其他选项之间也不能有空格,如%8.2f。

2. 常用格式字符的含义

常用字符串格式字符的说明如表 5.3 所示。

表 5.3 常用字符串格式字符的说明

格式字符	说　明
%c	格式化字符或编码
%s	格式化字符串
%d、%i	格式化整数

续表

格式字符	说　明
%u	格式化无符号整数
%%	百分号(%)
%o	格式化八进制数
%x	格式化十六进制数
%f	格式化浮点数,默认保留6位小数,可指定小数位数
%F	同%f;并且将inf和nan分别转换为INF和NAN
%e、%E	分别用e和E表示科学记数法格式的浮点数,如1.2e+03表示1.2×10^3
%g、%G	根据值的大小采用科学记数法或者浮点数形式;采用科学记数法时,分别用e和E表示;当数值中的数字个数大于6时,默认保留6个数字,可以自己指定保留的数字个数

3. 最小宽度和精度

最小宽度是转换后的值所保留的最小字符个数。精度(对于数字来说)则是结果中应该包含的小数位数。

```
>>> a = 3.1416
>>> '%6.2f' % a
'  3.14'
```

上述代码把a转换为含6个字符的小数串,保留2位小数,对第2位四舍五入。不足6个字符则在左边补空格。例如:

```
>>> '%f' % 3.1416                    #单独的%f默认保留6位小数
'3.141600'
>>> '%.2f' % 3.1416                  #指定保留2位小数
'3.14'
>>> '%7.2f' % 3.1416                 #宽度7位,保留2位小数,空位填空格
'   3.14'
>>> '%07.2f' % 3.1416                #宽度7位,保留2位小数,空位填0
'0003.14'
>>> '%+07.2f' % 3.1416               #宽度7位,保留2位小数,正数加正号,空位填0
'+003.14'
>>> '%-7.2f' % -3.1416               #宽度7位,保留2位小数,空位填空格,左对齐输出
'-3.14 '
>>> "%2d" % 56
'56'
>>> "%2d" % 5
' 5'
>>> "%-2d" % 56
'56'
>>> "%-2d" % 5
'5 '
```

上述代码中,"%-2d"%5表示5占两个字符宽度,左对齐输出。因此,输出中5后面补一个空格。

下面例子中,字符串'5'用格式化整数%d输出,引发异常。

```
>>> '%d' % '5'
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    '%d'%'5'
TypeError: %d format: a number is required, not str
>>> '%s' % 5                          #与 str()等价
'5'
```

可以一次格式化多个对象,这些对象表示成一个元组形式。这些对象的位置与格式化字符的位置一一对应。例如:

```
>>> '%.2f,%4d,%s' % (3.456727,89,'Lily')
'3.46, 89,Lily'
```

上述代码中,%.2f 表示 3.456727 的格式形式,%4d 表示 89 的格式形式,%s 表示'Lily'的格式形式,字符串里面的逗号(,)原封不动输出。

格式化运算符后面也可以是变量,例如:

```
>>> name = 'Lily'
>>> age = 18
>>> '我叫%s,今年%d岁' % (name,age)
'我叫 Lily,今年 18 岁'
```

4. 进位制和科学记数法

把一个数转换成不同的进位制,也可按科学记数法进行转换。例如:

```
>>> a = 123456
>>> y = '%o' % a                      #转换为八进制串
>>> y
'361100'
>>> ya = '%#o' % a                    #八进制时前添加 0o
>>> ya
'0o361100'
>>> z = '%x' % a                      #转换为十六进制串
>>> z
'1e240'
>>> za = '%#x' % a                    #十六进制数前添加 0x
>>> za
'0x1e240'
>>> se = '%e' % a                     #转换为科学记数法串,中间用 e
>>> se
'1.234560e+05'
```

以上代码表示将十进制数 a 分别转换为八进制串、十六进制串和科学记数法串。

如下代码演示了格式字符 e、E、f、F、g 和 G 的用法。

```
>>> '%e' % 12345.678                  #采用科学记数法形式,中间用 e
'1.234568e+04'
>>> '%E' % 12345.678                  #采用科学记数法形式,中间用 E
'1.234568E+04'
>>> '%F' % 12345.678
'12345.678000'
```

```
>>> '%f' % 12345.678
'12345.678000'
>>> '%g' % 12345.678          #采用浮点数形式,g或G默认都保留6个数字
'12345.7'
>>> '%g' % 1234.5678
'1234.57'
>>> '%g' % 123.45678
'123.457'
>>> '%g' % 12.345678
'12.3457'
>>> '%g' % 1.2345678
'1.23457'
>>> '%.3g' % 12345.678        #保留3个数字,自动转换为用e表示的科学记数法形式
'1.23e+04'
>>>
>>> '%G' % 12345.678          #采用浮点数形式,g或G默认保留6个数字
'12345.7'
>>> '%.4G' % 12345.678        #保留4个数字,自动转换为用E表示的科学记数法形式
'1.235E+04'
>>> '%.3G' % 12345.678        #保留3个数字,自动转换为用E表示的科学记数法形式
'1.23E+04'
>>>
>>> '%g' % 12345              #少于6个数字的整数,以实际位数显示整数
'12345'
>>> '%g' % 123456             #6个数字的整数,以实际位数显示整数
'123456'
>>> '%g' % 1234567            #多个6个数字的整数,转换为科学记数法表示,数字保留6位
'1.23457e+06'
>>>
```

【例 5.2】 利用格式化字符方式输出如图 5.2 所示的九九乘法表。

```
1*1=1
2*1=2   2*2=4
3*1=3   3*2=6   3*3=9
4*1=4   4*2=8   4*3=12  4*4=16
5*1=5   5*2=10  5*3=15  5*4=20  5*5=25
6*1=6   6*2=12  6*3=18  6*4=24  6*5=30  6*6=36
7*1=7   7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49
8*1=8   8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64
9*1=9   9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

图 5.2　九九乘法表

程序源代码如下：

```
#example5_2.py
#coding = utf - 8
for i in range(1,10):
    for j in range(1,i+1):
        print("%d*%d=%-4d"%(i,j,i*j),end=" ")
    print()
```

在例 5.2 中，乘数和被乘数均占一个字符的宽度输出；积占四个字符的宽度输出且左对齐。

5.3.2 用 format()方法格式化字符串

从 Python 2.6 开始加入了 format()方法来格式化字符串,使用起来更加方便和简洁。format()方法通过{}和:来代替传统%的方式来表示格式的定义。其一般形式如图 5.3 所示。

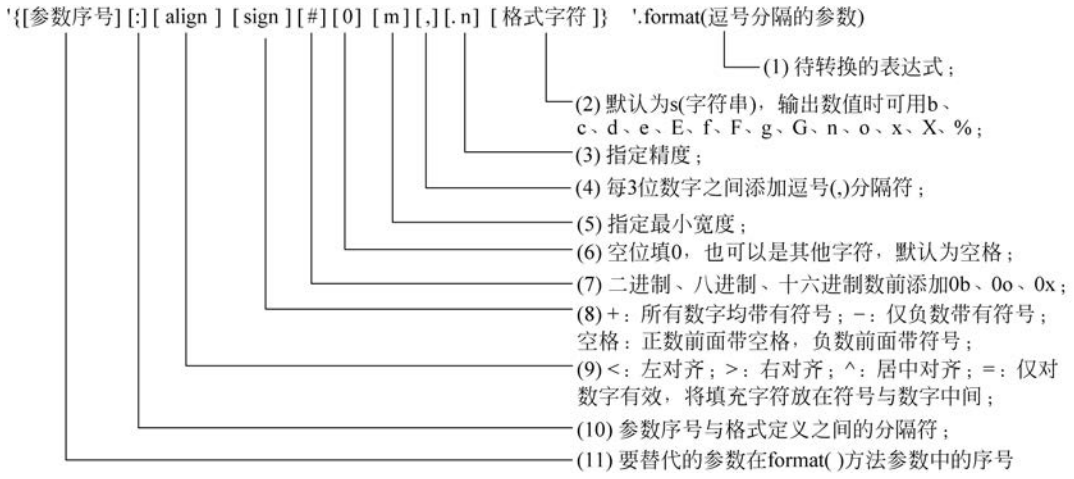

图 5.3 format()方法的一般形式

在一个字符串中可以有多个花括号{和}括起来的格式定义与占位符。format()方法中的大部分格式字符与传统的利用%进行格式化的格式字符相同。格式符 n 与 g 的功能相同,插入随区域而异的数字分隔符。格式符%表示将数字表示为百分数,也就是将参数值乘以 100,然后在后面加上百分号。

format()方法格式化时,可以使用位置参数,根据位置顺序来传递参数。例如:

```
>>> '我叫{},今年{}岁'.format('张清',18)
'我叫张清,今年 18 岁'
>>>
```

也可以通过索引值来引用位置参数,只要 format()方法相应位置上有参数值即可,参数索引从 0 开始。例如:

```
>>> '我叫{0},今年{1}岁'.format('张清',18)
'我叫张清,今年 18 岁'
>>> '我叫{1},今年{0}岁'.format(18,'张清')
'我叫张清,今年 18 岁'
>>>
```

也可以使用序列,通过 format()方法中序列参数的位置索引和序列中元素索引来引用相应值。例如:

```
>>> my = ['张清',18]
>>> '我叫{0[0]},今年{0[1]}岁'.format(my)
'我叫张清,今年 18 岁'
>>>
```

也可以用“*序列名称”的形式作为format()方法的实际参数。作为实际参数的序列名称前面加星号(*)表示将序列展开,然后通过位置依次将展开后的元素传递到目标字符串中。例如:

```
>>> '我叫{},今年{}岁'.format(*my)
'我叫张清,今年18岁'
>>>
```

也可以使用关键参数的形式(变量名=值)为format()方法传递实际参数。例如:

```
>>> '我叫{name},今年{age}岁'.format(name='张清',age=18)          #关键参数形式
'我叫张清,今年18岁'
>>> name='张清';age=18
>>> '我叫{name},今年{age}岁'.format(name,age)                    #不能这样写
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    '我叫{name},今年{age}岁'.format(name,age)
KeyError: 'name'
>>> '我叫{n},今年{a}岁'.format(n=name,a=age)                      #采用关键参数形式
'我叫张清,今年18岁'
>>>
```

也可用“**字典名”的形式为format()方法传递实际参数。作为实际参数的字典名称前面加两个星号(**)将字典中的元素依次展开为“键=值”的关键参数形式。例如:

```
>>> my={'name':'张清','age':18}
>>> '我叫{name},今年{age}岁'.format(**my)
'我叫张清,今年18岁'
>>>
```

位置参数与关键参数传递方式、序列前面加一个星号的参数传递方式、字典前面加两个星号的参数传递方式将在第6章详细介绍。

如果需要设置格式,则按照图5.3中的模式在冒号后面设置格式符。例如:

```
>>> '{0:.2f}'.format(2/3)
'0.67'
>>> '{0:b}'.format(8)                          #二进制
'1000'
>>> '{0:o}'.format(8)                          #八进制
'10'
>>> '{0:x}'.format(18)                         #十六进制
'12'
>>> '{0:#x}'.format(10)                        #十六进制前加0x
'0xa'
>>> '{:,}'.format(1234567890)                  #千分位格式化,数字每3位加一个逗号
'1,234,567,890'
>>> '{0:*>10}'.format(18)                      #右对齐
'********18'
>>> '{0:*<10}'.format(18)                      #左对齐
'18********'
>>> '{x:*^10}'.format(x=18)                    #居中对齐
'****18****'
```

```
>>> '{0:* =10}'.format(-18)                    # * 放在 - 和 18 中间
'- *******18'
>>> '{0:_},{0:#x}'.format(9999)                # _作为分隔符
'9_999,0x270f'
```

5.3.3 用 f-strings 字面量方法格式化字符串

Python 3.6 开始增加了 f-strings 特性,称为字面量格式化字符串。如果一个字符串前面带有 f 或 F 字符,则字符串中可以含有表达式,该表达式需要用花括号括起来。将计算完的花括号内的表达式结果转换为字符串,替换到该花括号及其内部表达式所在的位置,生成一个格式化的字符串对象。这种格式化方式类似于字符串的 format()方法,使用起来更加灵活、方便。推荐使用此方式进行字符串的格式化。

f-strings 采用 {content : format} 设置字符串格式。content 是替换并填入字符串的内容,可以是变量、表达式、函数、lambda 表达式等。format 是格式描述符,与字符串 format()方法中的格式描述符相同。采用默认格式时不必指定格式描述符,只需要{content}即可。例如:

```
>>> name = '张清'
>>> age = 18
>>> weight = 60.5
>>> s = F"我叫{name:>6s},今年{age:^4d}岁,体重{weight:.2f}kg。"
>>> s
'我叫  张清,今年 18 岁,体重 60.50kg。'
>>>
>>> import datetime
>>> birth = 1990
>>> high = 180
>>> s = f'我叫{name},今年{datetime.datetime.now().year-birth}岁,身高{high:.2f}cm。'
>>> s
'我叫张清,今年 29 岁,身高 180.00cm。'
>>>
```

格式描述符的详细用法可以参考 Python 官方文档(登录网址 https://docs.python.org,执行 Documentation→Library Reference→Text Processing Services→Format String Syntax 命令)。

花括号中可以放入系统内置函数、对象的方法或自定义函数。例如:

```
>>> s = f"姓名:{input('请输入姓名:')} 学号:{int(input('请输入学号:'))}"
请输入姓名:杨
请输入学号:2020055001
>>> s
'姓名:杨 学号:2020055001'
>>>
>>> s = f"转换为大写字母后:{input('英文字符串:').upper()}"
英文字符串:abcd
>>> s
'转换为大写字母后:ABCD'
>>>
>>> def fun(x):
```

```
    return x + 10

>>> s = f"函数调用结果:{fun(5)}"
>>> s
'函数调用结果:15'
>>>
```

上述代码中定义了一个自定义函数 fun()，然后在格式化字符串的花括号内调用了该函数，得到 5＋10 的结果。自定义函数的方法将在第 6 章详细介绍。

花括号中也可以放入列表和字典的相关运算。例如：

```
>>> list1 = [1,6,8,"a","b","c","d"]
>>> s = F"列表的切片:{list1[1:6:2]}"
>>> s
"列表的切片:[6, 'a', 'c']"
>>>
>>> d = {"a":1,"b":2}
>>> s = f'取字典中指定 key 的 value:{d["b"]}'
>>> s
'取字典中指定 key 的 value:2'
>>>
```

花括号中也可以是三元运算等其他表达式，将返回的结果替入相应位置。

5.4　字符串前加特定字符的作用

一个字符串前面可以带有 r、b 等字符，用来表示特定的含义。

5.4.1　去除转义功能

在普通字符串中，反斜杠用于和后面的字符一起构成表示特殊含义的转义字符，如'\n'表示换行、'\t'表示一个 Tab 键的位置等。这些字符串中的反斜杠和后面的字符共同表示一个特殊含义的字符。如果要使这些字符去掉转义的功能，如需要让字符串'\n'中的反斜杠\和字母 n 分别表示两个字符的原意，那么可以在字符串的开始边界符前面加上字符 r 或 R。例如：

```
>>> x = "a\nb"              #\n 构成一个转义字符,表示一个换行符
>>> len(x)
3
>>> print(x)
a
b
>>> y = r"a\nb"             #前面的 r 去掉了反斜杠的转义功能,\n 分别表示两个单独的字符
>>> len(y)
4
>>> print(y)
a\nb
>>>
```

5.4.2 字节串的表示

字符串前加 b 或 B 表示使用 ASCII 编码的字节串来表示字符串。这种方式只能对 ASCII 表中的字符所构成的字符串进行编码,字符串中不能包含非 ASCII 表中的字符。例如:

```
>>> y = b"我是学生"
SyntaxError: bytes can only contain ASCII literal characters.
>>> z = b"abc"
>>> z
b'abc'
>>> type(z)
<class 'bytes'>
>>>
```

可以使用 encode()方法将任何字符串转换为字节串;反过来,可以利用字节串的 decode()方法将任何字节串转换为字符串。5.1 节中已经介绍过这种方法,为了加深理解,这里再举一个相互转换的例子。

```
>>> s = z.decode("ascii")
>>> s
'abc'
>>> type(s)
<class 'str'>
>>>
```

5.1 节中提到,可以利用 bytes()的对象创建方法将任何字符串转换为字节串,也可以利用 str()的对象创建方法将任何字节串转换为字符串。

Python 3 中,对字符串默认采用 Unicode 编码。对于一些没有采用 Unicode 编码的字符串,前面加 u 可以强制字符串采用 Unicode 格式进行编码。


视频讲解

5.5 字符串截取

字符串的截取就是取出字符串中的子串。截取有两种方法:一种是索引 str[index]取出单个字符;另一种是切片 str[[start]:[end]:[step]]取出一片字符。切片方式与 4.1.4 节介绍的一样。

字符串中字符的索引与列表一样,可以双向索引,如图 5.4 所示。

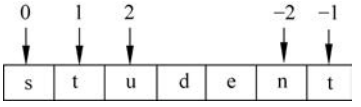

图 5.4 字符串中字符的索引位置,以字符串 'student' 为例

```
>>> s = 'student'
>>> s[0]
's'
>>> s[-1]
't'
>>> s[1:3]                    #取出位置为 1 到位置为 2 的字符,不包括位置为 3 的字符
'tu'
>>> s[:3]                     #取出从头至位置为 2 的字符
```

```
'stu'
>>> s[-2:]                    #取出从倒数第2个位置开始的所有字符
'nt'
>>> s[:]                      #取出全部字符
'student'
>>> s[::2]                    #步长为2
'suet'
>>> s[0]='e'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    s[0]='e'
TypeError: 'str' object does not support item assignment
>>> s[1:3]='ut'
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    s[1:3]='ut'
TypeError: 'str' object does not support item assignment
```

字符串属于不可变序列类型,不支持字符串修改。


视频讲解

【例5.3】 用import this可以导入Python之禅的文字描述。请用列表lists保存Python之禅的前4句,每句字符串作为列表的一个元素。从键盘分别输入表示子串开始和结束的位置的数字,在列表的每个字符串元素中分别截取两个位置之间的子串(假设第1个字符的位置为1;要取的子串包含开始和结束两个位置的字符)。采用字符串格式化形式分别输出列表中所有元素的字符串及其长度,以及相应位置之间的子串。这里假定两个位置的输入值均正确。

Python之禅的内容如下。

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
(略去剩余部分)
```

一种实现方法的程序源代码如下:

```
#example5_3.py
#coding=utf-8
lists=["Beautiful is better than ugly.","Explicit is better than implicit.",\
      "Simple is better than complex.","Complex is better than complicated."]
d1=int(input("请输入第一个位置:"))
d2=int(input("请输入第二个位置:"))
for s in lists:
    #print('字符串%s,长度为:%d,子串为:%s'%(s,len(s),s[d1-1:d2]))
    #print('字符串{},长度为:{},子串为:{}'.format(s,len(s),s[d1-1:d2]))
    print(f'字符串{s},长度为:{len(s)},子串为:{s[d1-1:d2]}')
```

程序 example5_3.py 的一次运行结果如下：

```
>>>
============ RESTART: G:\ example5_3.py ============
请输入第一个位置:3
请输入第二个位置:14
字符串 Beautiful is better than ugly.,长度为:30,子串为:autiful is b
字符串 Explicit is better than implicit.,长度为:33,子串为:plicit is be
字符串 Simple is better than complex.,长度为:30,子串为:mple is bett
字符串 Complex is better than complicated.,长度为:35,子串为:mplex is bet
```


视频讲解

5.6 字符串常用内置函数

在 Python 中有很多内置函数可以对字符串进行操作。如 len()、ord()、chr()、max()、min()等。例如：

```
>>> s = 'Merry days will come,believe.'
>>> len(s)                                  #字符串长度
29
>>> max(s)                                  #最大字符
'y'
>>> min(s)                                  #最小字符
' '
>>> ord('M')                                #获取该字符的 Unicode 码
77
>>> chr(77)                                 #把编码转换为对应的字符
'M'
>>> ord('好')
22909
>>> chr(22909)
'好'
```

【例 5.4】 请用两首歌曲名(《优美旅程》、*Fantasy*)组成列表 songs,编写程序,依次显示每首歌曲名、歌曲名的每个字符以及该字符的 Unicode 编码。

• 第一种方法。

程序源代码如下：

```
#example5_4_1.py
#coding = utf - 8
songs = ["优美旅程","Fantasy"]
for s in songs:
    print(s)
    for i in s:                             #直接遍历序列中的元素
        print(i,ord(i),sep = ' - ',end = ' ')
    print()
```

程序 example5_4_1.py 的一次运行结果如下：

```
>>>
============ RESTART: G:\ example5_4_1.py ============
```

```
优美旅程
优 - 20248 美 - 32654 旅 - 26053 程 - 31243
Fantasy
F - 70 a - 97 n - 110 t - 116 a - 97 s - 115 y - 121
```

- 第二种方法。

程序源代码如下：

```
#example5_4_2.py
#coding = utf - 8
songs = ["优美旅程","Fantasy"]
for s in songs:
    print(s)
    for i in range(len(s)):                    #通过序列索引遍历元素
        print(s[i],ord(s[i]),sep = ' - ',end = ' ')
    print()
```

字符串属于序列，与列表一样，可以有两种方法遍历序列。

思考 1：如果要求输出为如下形式，程序应该如何修改？

```
优美旅程
字符优的 Unicode 编码为 20248
字符美的 Unicode 编码为 32654
字符旅的 Unicode 编码为 26053
字符程的 Unicode 编码为 31243
Fantasy
字符 F 的 Unicode 编码为 70
字符 a 的 Unicode 编码为 97
字符 n 的 Unicode 编码为 110
字符 t 的 Unicode 编码为 116
字符 a 的 Unicode 编码为 97
字符 s 的 Unicode 编码为 115
字符 y 的 Unicode 编码为 121
```

修改的方法很多，其中一种修改方法的程序源代码如下：

```
#question5_4_1.py
#coding = utf - 8
songs = ["优美旅程","Fantasy"]
for s in songs:
    print(s)
    for i in s:
        print("字符 %s 的 Unicode 编码为 %d" %(i,ord(i)))
```

思考 2：如果只显示歌曲名字符串的下标为奇数的字符以及该字符的 Unicode 编码，程序应该如何修改？

修改后的程序源代码如下：

```
#question5_4_2.py
#coding = utf - 8
songs = ["优美旅程","Fantasy"]
for s in songs:
```

```
    print(s)
    for i in range(1,len(s),2):
        print("字符{}的 Unicode 编码为{}".format(s[i],ord(s[i])))
```

程序 question5_4_2.py 的运行结果如下：

```
>>>
============ RESTART: G:\ question5_4_2.py ============
优美旅程
字符美的 Unicode 编码为 32654
字符程的 Unicode 编码为 31243
Fantasy
字符 a 的 Unicode 编码为 97
字符 t 的 Unicode 编码为 116
字符 s 的 Unicode 编码为 115
```

【例 5.5】 编写程序，输入一个字符串，分别统计大写字母、小写字母、数字以及其他字符的个数，并分别以前面介绍的 3 种字符串格式化方式分别显示各种字符的个数。数字仅包括阿拉伯数字。

程序源代码如下：

```
#example5_5.py
#coding = utf - 8
s = input('请输入一个字符串:')
c1,c2,c3,c4 = 0,0,0,0
for i in s:
    if "A"<= i <= "Z":
        c1 += 1
    elif "a"<= i <= "z":
        c2 += 1
    elif "0"<= i <= "9":
        c3 += 1
    else:
        c4 += 1
print("大写字母 %d 个;小写字母 %d 个;数字 %d 个;其他字符 %d 个。" %(c1,c2,c3,c4))
print("大写字母{0}个;小写字母{1}个;数字{2}个;其他字符{3}个。".format(c1,c2,c3,c4))
print(f"大写字母{c1}个;小写字母{c2}个;数字{c3}个;其他字符{c4}个。")
```

程序 example5_5.py 的一次运行结果如下：

```
>>>
============ RESTART: G:\ example5_5.py ============
请输入一个字符串:学生 I am a student since 2000
大写字母 1 个;小写字母 15 个;数字 4 个;其他字符 8 个。
大写字母 1 个;小写字母 15 个;数字 4 个;其他字符 8 个。
大写字母 1 个;小写字母 15 个;数字 4 个;其他字符 8 个。
```


视频讲解

5.7 字符串常用方法

由于字符串属于不可变序列类型，常用方法中涉及返回字符串的都是新字符串，原有字符串对象不变。

1. center()、ljust()、rjust()

格式：center(self, width, fillchar = ' ', /)
　　ljust(self, width, fillchar = ' ', /)
　　rjust(self, width, fillchar = ' ', /)

说明：width 用于指定宽度；

　　fillchar 表示填充的字符，默认为空格。

功能：返回一个宽度为 width 的新字符串，原字符串居中(左对齐或右对齐)出现在新字符串中，如果 width 大于字符串长度，则使用 fillchar 进行填充。例如：

```
>>> '你好'.center(10)                    #居中对齐,以空格填充
    '你好 '
>>> '你好'.center(10,"*")                #居中对齐,以*填充
'****你好****'
>>> '你好'.ljust(10,"!")                 #右对齐,以!填充
'你好!!!!!!!!'
>>> '你好'.rjust(10,"-")                 #左对齐,以-填充
'--------你好'
```

【例 5.6】 使用 format()方法和 ljust()方法打印出例 5.2 中的九九乘法表。

程序源代码如下：

```
#example5_6.py
#coding = utf-8
for i in range(1,10):
    for j in range(1,i+1):
        print("{0}*{1}={2}".format(i,j,i*j).ljust(8),end=" ")
    print()
```

2. lower()、upper()


视频讲解

lower()方法将大写字母转换为小写字母，其他字符不变，并返回新字符串。upper()方法将小写字母转换为大写字母，其他字符不变，并返回新字符串。经常用这两种方法来解决不区分大小写的相关问题。例如：

```
>>> s = 'PYthon is A programming language.'
>>> s.lower()
'python is a programming language.'
>>> s                                    #原字符串不变
'PYthon is A programming language.'
>>> s1 = s.lower()
>>> s1                                   #新字符串
'python is a programming language.'
>>> s.upper()
'PYTHON IS A PROGRAMMING LANGUAGE.'
>>> s                                    #原字符串不变
'PYthon is A programming language.'
>>> s2 = s.upper()
>>> s2                                   #新字符串
'PYTHON IS A PROGRAMMING LANGUAGE.'
```

【例 5.7】 有一批淘宝用户名：风云 Th、Brown、飘然 12345、云 S、thomas。编写程序，将这些用户名存储在一个列表中，从键盘输入不区分大小写的用户名，判断能否找到该用户名。例如，输入 Thomas、THOmas、thoMAs 等均能找到 thomas。

程序源代码如下：

```
#example5_7.py
#coding = utf - 8
names = ["风云 Th","Brown","飘然 12345","云 S","thomas"]
name = input("请输入用户名:")
for s in names:
    if name.lower() == s.lower():
        print("找到")
        break
else:
    print("未找到")
```

程序 example5_7.py 的运行结果如下：

```
>>>
============ RESTART: G:\example5_7.py ============
请输入用户名:Thomas
找到
>>>
============ RESTART: G:\example5_7.py ============
请输入用户名:yhoms
未找到
>>>
============ RESTART: G:\example5_7.py ============
请输入用户名:风云 th
找到
>>>
```

【例 5.8】 用户从键盘依次输入若干字符串组成一个列表 list1。每输完一个字符串加入列表后，询问是否结束输入。如果此时输入 y 或者 yes(大小写无关)，则结束输入。然后将该列表转换为元组 tuple1，分别输出 list1 和 tuple1。

程序源代码如下：

```
#example5_8.py
#coding = utf - 8
print("请输入若干字符串组成列表 list1")
yy = 'n'
i = 1
list1 = []                                  #初始化一个空列表
while yy.upper() not in ['Y','YES'] :       #判断是否结束
    x = input("请输入第" + str(i) + "个元素:")
    list1.append(x)
    i += 1
    yy = input("输入结束了吗?(y 或 yes 表示结束,大小写无关,其他继续):")
tuple1 = tuple(list1)
print("列表 list1:",list1)
print("元组 tuple1:",tuple1)
```

程序 example5_8.py 可能的一次运行结果如下：

```
>>>
 ============ RESTART: G:\example5_8.py ============
请输入若干字符串组成列表 list1
请输入第 1 个元素:Alice
输入结束了吗?(y 或 yes 表示结束,大小写无关,其他继续):n
请输入第 2 个元素:Tom
输入结束了吗?(y 或 yes 表示结束,大小写无关,其他继续):ye
请输入第 3 个元素:Rose
输入结束了吗?(y 或 yes 表示结束,大小写无关,其他继续):y
列表 list1: ['Alice', 'Tom', 'Rose']
元组 tuple1: ('Alice', 'Tom', 'Rose')
```

3. capitalize()、title()、swapcase()


视频讲解

capitalize()方法将字符串首字母转换为大写形式，其他字母转换为小写形式。title()方法将每个单词的首字母转换为大写形式，其他部分的字母转换为小写形式。swapcase()字符将大小写互换。三种方法均返回新字符串，原字符串对象不进行任何修改。例如：

```
>>> s = 'merry days will come, believe.'
>>> s.capitalize()
'Merry days will come, believe.'
>>> s1 = s.title()
>>> s1
'Merry Days Will Come, Believe.'
>>> s
'merry days will come, believe.'
>>> s1.swapcase()
'mERRY dAYS wILL cOME, bELIEVE.'
```

4. islower()、isupper()、isdigit()


视频讲解

islower()、isupper()、isdigit()方法的功能分别是测试字符串是否为小写字母、大写字母、数字字符。如果是，则返回 True；否则返回 False。例如：

```
>>> s = 'merry days will come, believe.'
>>> s.islower()
True
>>> s.isupper()
False
>>> s = '1234'
>>> s.isdigit()
True
>>> s = '1234.5'
>>> s.isdigit()
False
```

str 类中还有一些测试字符串是否为空白字符的方法，请读者利用 help(str)命令自行查看帮助信息。

【例 5.9】 改写例 5.5 的程序，利用 isupper()、islower()、isdigit()方法分别判断是否为大写字母、小写字母、数字字符，并分别统计以上三种字符的个数。用字符串格式化方式

分别显示三种字符的个数。

程序源代码如下：

```
#example5_9.py
#coding = utf - 8
s = input('请输入一个字符串:')
c1,c2,c3,c4 = 0,0,0,0
for i in s:
    if i.isupper():
        c1 += 1
    elif i.islower():
        c2 += 1
    elif i.isdigit():
        c3 += 1
    else:
        c4 += 1
print(f"大写字母{c1}个;小写字母{c2}个;数字字符{c3}个;其他字符{c4}个。")
```

程序 example5_9.py 的一次运行结果如下：

```
>>>
============ RESTART: G:\ example5_9.py ============
请输入一个字符串:学生 I am a student since 2000
大写字母 1 个;小写字母 15 个;数字字符 4 个;其他字符 8 个。
```


视频讲解

5. find()、rfind()

格式：s.find(sub[,start[,end]])

s. rfind(sub[,start[,end]])

说明：sub 表示字符串(子串)；

start 表示开始位置；

end 表示结束位置。查找范围从 start 开始到 end 结束，不包括 end。

功能：在一个较长的字符串 s 中，在[start,end)范围内查找并返回子串 sub 首次出现的位置索引，如果没有找到则返回－1。默认范围是整个字符串。其中 find()方法为从左往右查找，rfind()方法为从右往左查找。例如：

```
>>> s = 'Heart is living in tomorrow'
>>> s.find('Heart')                    #在整个字符串范围内查找
0
>>> s.find('is')
6
>>> s.find('heart')                    #子串不存在返回 - 1
-1
>>> s.find('i')                        #有多个子串'i',只会返回查找范围内第 1 次出现的位置
6
>>> s.find('i',7)                      #从索引位置 7 开始查找到最后
10
>>> s.find('i',11)
12
>>> s.find('i',13)
16
```

```
>>> s.find('i',17)
-1
>>> s.find('i',11,16)                    #从索引位置 11 开始查找到索引位置 16(不包括 16)
12
>>> s.find('i',7,10)
-1
>>> s.rfind('i')                         #返回最右端索引位置
16
>>> s.rfind('i',0,16)
12
```

6. index()、rindex()


视频讲解

格式：s.index(sub[,start[,end]])

　　s. rindex(sub[,start[,end]])

功能：在一个较长的字符串 s 中，查找并返回在[start,end)范围内子串 sub 首次出现的位置索引，如果不存在则抛出异常。默认范围是整个字符串。其中 index()方法为从左往右查找，rindex()方法为从右往左查找。例如：

```
>>> s = 'Heart is living in tomorrow'
>>> s.index('i')
6
>>> s.index('i',7)
10
>>> s.index('is')
6
>>> s.index('in')
12
>>> s.index('heart')                     #子串不存在,抛出异常
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    s.index('heart')
ValueError: substring not found
>>> s.index('i',7,11)
10
>>> s.index('i',7,10)                    #[7,10)范围内子串不存在,抛出异常
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    s.index('i',7,10)
ValueError: substring not found
>>> s.rindex('i')                        #字符'i'在字符串 s 中最后一次出现的位置
16
```

7. count()


视频讲解

格式：s.count(sub[,start[,end]])

功能：在一个较长的字符串 s 中，查找并返回[start,end)范围内子串 sub 出现的次数，如果不存在则返回 0。默认范围是整个字符串。例如：

```
>>> s = 'Heart is living in tomorrow'
>>> s.count('i')
4
```

```
>>> s.count('i',11)
2
>>> s.count('Is')
0
```


视频讲解

8. split()方法

split(self,/,sep=None,maxsplit=-1)方法以 sep 指定字符为分隔符,从左往右将字符串分隔开来,并将分隔后的子串组成列表返回。参数 maxsplit 表示最大分隔次数;默认为-1,表示分隔次数没有限制,只要出现分隔符 sep 就进行分隔。

```
>>> s1 = 'Heart,is,living,in,tomorrow'
>>> s1.split(",")                          #通过逗号","分隔
['Heart', 'is', 'living', 'in', 'tomorrow']
>>> s1.split(";")              #通过分号";"分隔;s1 中没出现分号,s1 整体作为列表的单一元素
['Heart,is,living,in,tomorrow']
>>> s2 = 'Heart is living in tomorrow'
>>> s2.split()                             #默认通过空白符分隔
['Heart', 'is', 'living', 'in', 'tomorrow']
>>> s2.split(',')
['Heart is living in tomorrow']
>>> s3 = 'Heart\tis\n\nliving\t\tin tomorrow'
>>> s3.split()                             #默认通过空白字符分隔
['Heart', 'is', 'living', 'in', 'tomorrow']
```

对于 split()方法,如果不指定分隔符,实际上表示以任何空白字符(包括连续出现的)作为分隔符。空白字符包括空格、换行符、制表符等。

除了 split()方法,还有 rsplit()方法,表示从右往左将字符串分隔开,这两种方法均能指定最大分隔次数。读者可以通过 help(str.rsplit)命令查看帮助信息,这里不再深入阐述。

【例 5.10】 编写程序,从键盘输入最近几天的温度,用逗号分隔,求平均温度,保留两位小数。

第一种方法的程序源代码如下:

```
#example5_10_1.py
#coding = utf - 8
s = input("请输入最近几天的温度,用逗号分隔:")
li = s.split(",")
ss = 0
for i in li:
    ss += float(i)

avg = ss/len(li)
print("平均温度:{:.2f}".format(avg))
```

程序 example5_10_1.py 的一次运行结果如下:

```
>>>
============ RESTART: G:\ example5_10_1.py ============
请输入最近几天的温度,用逗号分隔:25,27.5,29
平均温度:27.17
```

第二种方法的程序源代码如下：

```
#example5_10_2.py
#coding = utf - 8
s = input("请输入最近几天的温度,用逗号分隔:")
li = s.split(",")
c = list(map(float,li))
avg = sum(c)/len(c)
print("平均温度:{:.2f}".format(avg))
```

第三种方法的程序源代码如下：

```
#example5_10_3.py
#coding = utf - 8
c = eval(input("请输入最近几天的温度,用逗号分隔:"))
avg = sum(c)/len(c)
print("平均温度:{:.2f}".format(avg))
```

9. join()

join()方法可用来连接可迭代对象中的元素，并在两个元素之间插入指定字符串，返回一个字符串。例如：

```
>>> s1 = 'Heart is living in tomorrow'              #字符串序列中,每个字符为一个元素
>>> '+'.join(s1)
'H+e+a+r+t+ +i+s+ +l+i+v+i+n+g+ +i+n+ +t+o+m+o+r+r+o+w'
>>> s2 = ['Heart','is','living','in','tomorrow']   #该列表中,每个字符串为一个元素
>>> '+'.join(s2)
'Heart+is+living+in+tomorrow'
>>> s3 = 'Heart','is','living','in','tomorrow'     #s3 是元组,每个字符串为一个元素
>>> ','.join(s3)
'Heart,is,living,in,tomorrow'
>>> type(s3)
<class 'tuple'>
```

join()方法是 split()方法的逆方法。例如：

```
>>> s = 'Heart is living in tomorrow'
>>> slie = s.split()
>>> slie
['Heart', 'is', 'living', 'in', 'tomorrow']
>>> ss = ' '.join(slie)
>>> ss
'Heart is living in tomorrow'
```

【例 5.11】 编写程序，以空格分隔输入英文单词，然后将英文单词首字母大写、其他位置上的字母小写后输出。

程序源代码如下：

```
#example5_11_1.py
#coding = utf - 8
s = input('请输入英文单词,用空格分隔:')
w = s.split()
```

```
f = [i.title() for i in w]
ss = ' '.join(f)
print('单词首字母大写:',ss)
```

程序 example5_11_1.py 的一次运行结果如下：

```
>>>
============ RESTART: G:\ example5_11_1.py ============
请输入英文单词,用空格分隔:heart is living in tomorrow
单词首字母大写: Heart Is Living In Tomorrow
```

也可以将输入的整体作为字符串，直接利用字符串的 title()方法来实现，程序源代码如下：

```
#example5_11_2.py
#coding = utf - 8
s = input('请输入英文单词,用空格分隔:')
t = s.title()
print('单词首字母大写:',t)
```

10. replace()

replace(old,new,count=－1)方法用于查找字符串中的 old 子串并用 new 子串来替换。参数 count 默认值为－1，表示替换所有匹配项，否则最多替换 count 次。返回替换后的新字符串。例如：

```
>>> s = 'Heart is living in tomorrow'
>>> s.replace('i','I')
'Heart Is lIvIng In tomorrow'
>>> s                                          #原字符串不变
'Heart is living in tomorrow'
>>> s1 = '中国北京,北京地铁,地铁沿线,沿线站名'
>>> s2 = s1.replace('北京','Beijing')          #替换所有匹配项
>>> s2
'中国 Beijing,Beijing 地铁,地铁沿线,沿线站名'
>>> s3 = s1.replace('北上','Beijing')          #字符串中无"北上"匹配项
>>> s3
'中国北京,北京地铁,地铁沿线,沿线站名'
>>> s1 = '中国北京,北京地铁,地铁沿线,沿线站名'
>>> s4 = s1.replace('北京','Beijing',1)        #指定最大替换次数
>>> s4
'中国 Beijing,北京地铁,地铁沿线,沿线站名'
>>>
```

11. maketrans()、translate()

maketrans()方法用于生成字符映射表，translate()方法用于根据字符映射表替换字符。这两种方法联合起来使用可以一次替换多个字符。例如：

```
>>> t = ''.maketrans('iort','mn24')            #两个序列中的元素按照次序一一对应用于替换
>>> s = 'Heart is living in tomorrow'
>>> s.translate(t)
'Hea24 ms lmvmng mn 4nmn22nw'
```

【例 5.12】 编写程序，生成一个包含 10 个不重复的取自 a～z(随机生成)的小写字母的列表，将原列表中的 abcdefg 分别替换为 1234567。先输出原列表和新列表，再逐个输出新列表中的元素。

提示：产生随机数需要导入 random 模块，其中 random.randint(a,b)用于生成一个指定范围内的整数。其中参数 a 是下限，参数 b 是上限，生成的随机数 n 满足 a≤n≤b。

程序源代码如下：

```
#example5_12.py
#coding = utf - 8
import random
list1 = []

while len(list1)< 10:
   c = chr(random.randint(ord('a'),ord('z')))
   if c not in list1:
      list1.append(c)

print("原列表:",list1)
s1 = ','.join(list1)
t = ''.maketrans('abcdefg','1234567')
s2 = s1.translate(t)
list2 = s2.split(',')
print("新列表:",list2)
print("逐个输出新列表中的元素:")
for i in list2:
    print(i, end = ' ')
```

程序 example5_12.py 的一次运行结果如下：

```
>>>
============ RESTART: G:\ example5_12.py ============
原列表: ['x', 'e', 'b', 'c', 'j', 'w', 'a', 'l', 'p', 'r']
新列表: ['x', '5', '2', '3', 'j', 'w', '1', 'l', 'p', 'r']
逐个输出新列表中的元素:
x 5 2 3 j w 1 l p r
```

12. strip()、lstrip()、rstrip()

strip(chars=None,/)方法用于去除字符串两侧的空白字符或指定字符序列中的字符，并返回新字符串。lstrip(chars=None,/)方法用于去除字符串左侧的空白字符或指定字符序列中的字符，并返回新字符串。rstrip(chars=None,/)方法用于去除字符串右侧的空白字符或指定字符序列中的字符，并返回新字符串。例如：

```
>>> s = ' Heart is living in tomorrow \n \n'
>>> s.strip()            #没有指定字符参数,默认去除 s 两端的空白字符
'Heart is living in tomorrow'
>>> s1 = 'HHwHeart is liwving iHn tomorrowHww'
>>> s1.strip('Hw')       #从两端逐一去除字符序列'Hw'中的字符,直到不是该序列中的字符为止
'eart is liwving iHn tomorro'
>>> s1.lstrip('Hw')      #从左侧逐一去除字符序列'Hw'中的字符,直到不是该序列中的字符为止
```

```
'eart is liwving iHn tomorrowHww'
>>> s1.rstrip('Hw')      #从右侧逐一去除字符序列'Hw'中的字符,直到不是该序列中的字符为止
'HHwHeart is liwving iHn tomorro'
>>>
```

思考：如何去除字符串中所有的空格？

程序源代码如下：

```
>>> s = 'Heart is living in tomorrow'
>>> s.replace(' ','')    #将空格替换为空字符串
'Heartislivingintomorrow'
```

5.8 字符串 string 模块

字符串 string 模块定义了 Formatter 类、Template 类、capwords 函数和常量，熟悉 string 模块可以简化某些字符串的操作。可以通过 help()函数了解 string 模块的主要内容：

```
>>> import string
>>> help(string)
Help on module string:

NAME
    string - A collection of string constants.
...
FUNCTIONS
    capwords(s, sep = None)
        capwords(s [,sep]) -> string
       ...

DATA
    __all__ = ['ascii_letters', 'ascii_lowercase', 'ascii_uppercase', 'cap...
    ascii_letters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
    ascii_lowercase = 'abcdefghijklmnopqrstuvwxyz'
    ascii_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    digits = '0123456789'
    hexdigits = '0123456789abcdefABCDEF'
    octdigits = '01234567'
    printable = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTU...
    punctuation = '!"# $ %&\'()* +,-./:;< = >?@[\\]^_{|}~'
    whitespace = '\t\n\r\x0b\x0c'
```

【例 5.13】 输入由英文单词组成的字符串，单词之间用空格分隔，调用 string 模块中的 capwords()函数把字符串中每个英语单词的首字母转换为大写，其他字母转换为小写，返回新字符串。

程序源代码如下：

```
#example5_13.py
#coding = utf - 8
import string
s = input('请输入英文单词,用空格分隔:')
```

```
ss = string.capwords(s)
print('单词首字母大写:',ss)
```

程序 example5_13.py 的某一次运行结果如下：

```
请输入英文单词,用空格分隔:I am a student
单词首字母大写: I Am A Student
```

【例 5.14】 在例 5.12 中生成一个包含 10 个不重复的取自 a～z(随机生成)的小写字母的列表是利用 chr()函数和 random 模块中 randint()方法生成的。请改用 random.choice()函数从 string.ascii_lowercase 字符串常量中随机选择字符的方式实现随机生成 a～z 字符的功能。

程序源代码如下：

```
#example5_14.py
#coding = utf - 8
import random
import string
list1 = []
i = 0
while i < 10:
    c = random.choice(string.ascii_lowercase)
    if c not in list1:
        i += 1
        list1.append(c)
print("原列表:",list1)
```

程序 example5_14.py 的一次运行结果如下：

```
>>>
============ RESTART: G:\ example5_14.py ============
原列表: ['y', 'u', 's', 'l', 'x', 't', 'e', 'o', 'n', 'w']
```

说明：这种方法直接用到 string 模块中的常量 ascii_lowercase 和 random 模块中 choice()函数。choice()函数的功能是在一个非空的序列中随机选择一个元素。例如：

```
>>> import random
>>> x = '012345abcde'
>>> random.choice(x)
'd'
>>> random.choice(x)
'3'
>>> random.choice(x)
'1'
```

5.9　正则表达式

正则表达式是一个特殊的字符序列，利用事先定义好的一些特殊字符以及它们的组合组成一个规则(模式)，通过检查一个字符串是否与这种规则匹配来实现对字符的过滤或匹配。这些特殊的字符称为元字符。正则表达式是字符串处理的有力工具。

Python中,re模块提供了正则表达式操作所需要的功能。re模块中findall(pattern,string,flags=0)函数返回字符串string中所有非重叠匹配项(子串)构成的列表;如果没有找到匹配的,则返回空列表。pattern是正则表达式。string表示待匹配的字符串。flags是标志位,表示匹配方式,如是否忽略大小写、是否多行匹配等,这里不展开讨论。re模块中的split(pattern,string,maxsplit=0,flags=0)函数提供了按照正则表达式来切分字符串的功能,返回一个由各子串组成的列表。

普通字符会和自身匹配。例如:

```
>>> import re
>>> s = r'abc'
>>> re.findall(s,'aabaab')                    #无匹配
[]
>>> re.findall(s,'aabcaabc')                  #两处匹配
['abc', 'abc']
```

前面已经提到,在字符串前面加r或R表示去掉字符串内部的转义功能,保持原生字符,不进行转义。正则表达式字符串前一般写上r或R。除非正则表达式字符串内部确实有用转义方式表示的特殊字符,此时前面一定不能加r或R。

下面介绍常用的正则表达式元字符。

1. ".":表示除换行符以外的任意一个字符

```
>>> import re
>>> s = 'hi,i am a student.my name is Hilton.'
>>> re.findall(r'i',s)                        #匹配所有的i
['i', 'i', 'i', 'i']
>>> re.findall(r'.',s)                        #匹配除换行符以外的任意一个字符
['h', 'i', ',', 'i', ' ', 'a', 'm', ' ', 'a', ' ', 's', 't', 'u', 'd', 'e', 'n', 't', '.', 'm', 'y', ' ', 'n',
'a', 'm', 'e', ' ', 'i', 's', ' ', 'H', 'i', 'l', 't', 'o', 'n', '.']
>>> re.findall(r'i.',s)                       #匹配i后面跟除换行符以外的任意一个字符的形式
['i,', 'i ', 'is', 'il']
```

与"."类似(但不相同)的一个符号是"\S",表示不是空白符的任意一个字符。注意是大写字符S。例如:

```
>>> re.findall(r'i\S',s)                      #匹配i后面跟不是空白符的任意一个字符的形式
['i,', 'is', 'il']
```

2. "[]":指定字符集

(1)"[]"常用来指定一个字符集,如[abc]、[a-z]、[0-9]。

(2)元字符在方括号中不起作用,只表示本来的字符含义。例如,[akm$]和[m.]中元字符$和点(.)都只表示字符的原意,不起元字符的作用。

(3)方括号内的"^"表示补集,匹配不在区间范围内的字符,例如,[^3]表示除3以外的字符。

举例如下:

```
>>> import re
>>> s = 'map mit mee mwt meqwt'
>>> re.findall(r'me',s)
```

```
['me', 'me']
>>> re.findall(r'm[iw]t',s)              #匹配 m 后跟 i 或者 w 再跟 t 的形式
['mit', 'mwt']
>>> re.findall(r'm[.]',s)                #"."放在[]内,表示普通字符,不表示元字符
[]
>>> s = '0x12x3x567x8xy'
>>> re.findall(r'x[0123456789]x',s)      #每次只取[]中的一个字符
['x3x', 'x8x']
>>> re.findall(r'x[0-9]x',s)             #[0-9]与[0123456789]等价
['x3x', 'x8x']
>>> re.findall(r'x[^3]x',s)              #x 后跟不为 3 的字符再跟 x
['x8x']
```

3. “^”：匹配行首，匹配以^后面的字符开头的字符串

```
>>> import re
>>> s1 = "hello world, hello Mary."
>>> re.findall(r"hello", s1)             #匹配所有 hello 字符串
['hello', 'hello']
>>> re.findall(r"^hello", s1)            #匹配以 hello 开头的字符串
['hello']
>>> s2 = "hi world, hello Mary."
>>> re.findall(r"^hello", s2)            #s2 中没有以 hello 开头的字符串
[]
```

4. “$”：匹配行尾，匹配以 $ 之前的字符结束的字符串

```
>>> import re
>>> s = 'hello hello world hello Mary hello John'
>>> re.findall(r'hello$',s)
[]
>>> s = 'hello hello world hello Mary hello'
>>> re.findall(r'hello$',s)
['hello']
>>> s = 'map mit mee mwt meqmtm$'
>>> re.findall(r'm[aiw]$',s)             #匹配以 ma、mi、mw 结尾的字符串
[]
>>> re.findall(r'm[aiwt$]',s)            #$在[]中作为普通字符
['ma', 'mi', 'mw', 'mt', 'm$']
>>> re.findall(r'm[aiwt$]$',s)           #匹配以 ma、mi、mw、mt、m$ 结尾的字符串
['m$']
```

5. “\”：反斜杠后面可以加不同的字符以表示不同的特殊意义

(1) \b 匹配单词头或单词尾。

(2) \B 与\b 相反，匹配非单词头或单词尾。

(3) \d 匹配任何十进制数；相当于[0-9]。

(4) \D 与\d 相反，匹配任何非数字字符，相当于[^0-9]。

(5) \s 匹配任何空白字符，相当于[\t\n\r\f\v]。

(6) \S 与\s 相反，匹配任何非空白字符，相当于[^\t\n\r\f\v]。

(7) \w 匹配任何字母、数字或下画线字符，相当于[a-zA-Z0-9_]。

(8) \W 与\w 相反,匹配任何非字母、数字和下画线字符,相当于[^a-zA-Z0-9_]。

(9) 也可以放在元字符前用于取消元字符的功能。例如,“\\”和“\[”分别取消了反斜杠和左方括号的元字符功能,使右侧的反斜杠和左方括号只表示普通字符。

这些特殊字符都可以包含在[]中。如[\s,.]将匹配任何空白字符、","或"."。

```
>>> import re
>>> s = '0x12x3x567x8xy'
>>> re.findall(r'[0-9]',s)              #匹配 0～9 中的单个数字字符
['0', '1', '2', '3', '5', '6', '7', '8']
>>> re.findall(r'\d',s)
['0', '1', '2', '3', '5', '6', '7', '8']
>>> re.findall(r'[x\d]',s)              #匹配字母 x 或数字
['0', 'x', '1', '2', 'x', '3', 'x', '5', '6', '7', 'x', '8', 'x']
```

正则表达式除了能够匹配不定长的字符集,还能指定正则表达式的一部分的重复次数,所涉及的元字符有“*”“+”“?”“{}”。

6.“*”:匹配位于*之前的字符或子模式的0次或多次出现

```
>>> import re
>>> s = 'a ab abbbbb abbbbbxa'
>>> re.findall(r'ab*',s)                #a 后面跟重复 0 到多次的 b
['a', 'ab', 'abbbbb', 'abbbbb', 'a']
```

7.“+”:匹配位于+之前的字符或子模式的1次或多次出现

```
>>> import re
>>> s = 'a ab abbbbb abbbbbxa'
>>> re.findall(r'ab+',s)                #a 后面跟重复 1 到多次的 b
['ab', 'abbbbb', 'abbbbb']
```

8.“?”:匹配位于? 之前的0个或1个字符

当“?”紧随于其他限定符(*、+、{n}、{n,}、{n,m})之后时,匹配模式是“非贪心的”。“非贪心的”模式匹配搜索到尽可能短的字符串,而默认的“贪心的”模式匹配搜索到尽可能长的字符串。例如:

```
>>> import re
>>> s = 'a ab abbbbb abbbbbxa'
>>> re.findall(r'ab+',s)                #最大模式、贪心模式
['ab', 'abbbbb', 'abbbbb']
>>> re.findall(r'ab+?',s)               #最小模式、非贪心模式
['ab', 'ab', 'ab']
```

如果有字符串 s='hi,i am a student. my name is Hilton. ',那么 re.findall(r'i.*e',s)和 re.findall(r'i.*? e',s)会得到怎样不同的结果,为什么?

```
>>> import re
>>> s = 'hi,i am a student. my name is Hilton. '
>>> re.findall(r'i.*e',s)               #贪心模式
['i,i am a student. my name']
>>> re.findall(r'i.*?e',s)              #非贪心模式
['i,i am a stude']
```

在正则表达式中，“.”表示除换行符之外的任意字符，“*”表示位于它之前的字符可以重复任意0次或多次，只要满足这样的条件，都会被匹配。所以r'i.*e'表示i后面跟0个或多个除换行符之外的任意字符(最大模式匹配，贪心的)再跟字母e。那么re.findall(r'i.*e',s)会一直匹配到name。r'i.*?e'表示i后面跟0个或多个除换行符之外的任意字符，后面的“?”表示最小模式匹配(非贪心的)，然后再跟字母e。所以re.findall(r'i.*?e',s)会搜索到尽可能短的字符串，直到stude就结束了。

9. “{m,n}”：表示至少有m个重复，至多有n个重复。m、n均为十进制数

省略m表示0个重复，省略n表示无穷多个重复。{0,}等同于*；{1,}等同于+；{0,1}与?相同。如果可以，最好使用*、+、或?。例如：

```
>>> import re
>>> s = 'a b baaaaba'
>>> re.findall(r'a{1,3}',s)
['a', 'aaa', 'a', 'a']
>>> s = '021-33507yyx,021-33507865,010-12345678,021-123456789'
>>> re.findall(r'021-\d{8}',s)
['021-33507865', '021-12345678']
>>> re.findall(r'\b021-\d{8}\b',s)    #\b表示匹配字符串的头或尾
['021-33507865']
```

注意，因为“\b”是一个特殊符号的转义表示，为了使“\b”表示正则表达式中的元字符，必须在正则表达式的字符串前加r或R，去掉转义功能。

再来看一下上例中正则表达式字符串前如果不添加r或R的运行结果。

```
>>> re.findall("\b021-\d{8}\b",s)
[]
```

这个例子表示电话号码形式的前后均为转义字符“\b”表示的符号。因此字符串s中没有匹配的子串。

【例5.15】 随机产生10个长度为1～25，由字母、数字和“_”“.”“#”“%”特殊字符组成的字符串构成的列表，找出列表中符合下列要求的字符串：长度为5～20，必须以字母开头、可带数字和“_”“.”。

程序源代码如下：

```
#example5_15.py
#coding = utf-8
import string
import random
import re
z = []
#生成包含大小写字母、数字和指定符号的字符串
x = string.ascii_letters + string.digits + "_.#%"

#为了生成10个字符串元素，执行10次循环
for i in range(10):
    #生成字符作为元素，个数为1～25随机数的字符列表y
    y = [random.choice(x) for i in range(random.randint(1,25))]
    #用join()方法将字符列表y中的字符元素合并为字符串
    #并将此字符串加入列表z中
```

```
        z.append(''.join(y))

print("列表:",z)
print("满足要求的字符串是:")

#总长度为5～20
#以字母开头(1个字符):^[a-zA-Z]{1}
#可带"数字""_"".",至尾部共4～19个:[a-zA-Z0-9._]{4,19}$
m=r'^[a-zA-Z]{1}[a-zA-Z0-9._]{4,19}$'
for s in z:
    if re.findall(m,s):
        print(s)
```

程序 example5_15.py 的一次运行结果如下：

```
>>>
============ RESTART: G:\example5_15.py ============
列表:['uG5fSOIpLRcnJFZw2pqe5KEn', 'eO#.E%w9OCatpt0mZTXnV1V', 'AdIgyXk9lHNnfXdKW_cGNSrf2',
'S8', 'v4YSGIRODHmjlq', 'wRhB7_TfuZHGAI8U7VJ', 'rS6na8xdD6jhglHzflH4', 'S7UhrUhokQfJKsGJee',
'n4PTNb', 'YDw6qfdYswPzrzGCwn7t']
满足要求的字符串是:
v4YSGIRODHmjlq
wRhB7_TfuZHGAI8U7VJ
rS6na8xdD6jhglHzflH4
S7UhrUhokQfJKsGJee
n4PTNb
YDw6qfdYswPzrzGCwn7t
```

本节以 re 模块中的 findall()函数为例，介绍了正则表达式的用法。re 模块中还有很多其他非常有用的函数和类。读者可以通过帮助文档或其他资料来进一步深入了解它们的用法。

习题 5

1. 编写程序，输入一个时间(h：min：s)，输出该时间经过 5min30s 后的时间。

2. 编写程序，输入一个字符串，将该字符串中下标(索引)为偶数的字符组成新串并通过字符串格式化方式显示。

3. 编写程序，生成一个由 15 个不重复的大小写字母组成的列表。

4. 给定字符串"site sea suede sweet see kase sse ssee loses"，编写程序，匹配出所有以 s 开头、e 结尾的单词。

5. 编写程序，生成 15 个包括 10 个字符的随机密码，密码中的字符只能由大小写字母、数字字符和特殊字符"@""$""#""&""_""～"构成。

6. 给定列表 x=["13915556234","13025621456","15325645124","15202362459"]，编写程序，检查列表中的元素是否为移动手机号码，这里移动手机号码的规则是：手机号码共 11 位数字；以 13 开头，后面跟 4、5、6、7、8、9 中的某一个；或者以 15 开头，后面跟 0、1、2、8、9 中的某一个。