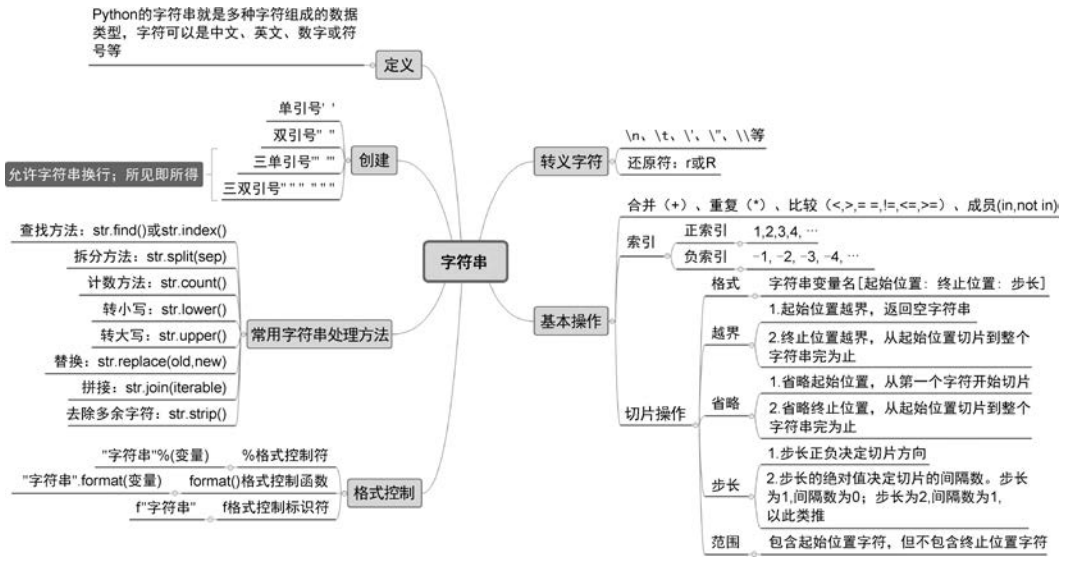
第3章 字符串

学习目标

- 掌握字符串创建的方法。
- 理解重点转义字符。
- 掌握字符串格式化方法。
- 熟练掌握字符串的基本操作。
- 掌握字符串的处理函数和方法。



在第 2 章已经介绍过数据类型，即六个标准的数据类型——数字类型、字符串类型、列表类型、元组类型、字典类型和集合类型。其中数字类型和字符串类型是基础类型，需要非常熟练地掌握字符串类型，为后面的四种复合数据类型打下基础。本章将详细介绍字符串类型，包括字符串的创建、字符串格式化方法、字符串的基本操作、字符串的处理函数和方法。

什么是字符串呢？字符串简单的理解就是由各种字符依次组成的字符集合，其中字符可以是数字、大小写字母、符号和汉字等，如"Hello,Charlie"是一个字符串、"123456"或"你好,Python。"也是一个字符串。Python 要求字符串必须使用引号括起来，而且两边的引号成对出现。

3.1 字符串的创建

3.1.1 定义字符串

字符串(String)就是若干个字符的集合。Python 中的字符串创建的方法有三种。

(1) 使用单引号包含字符，例如，'abc'、'123'。

(2) 使用双引号包含字符，例如，"abc"、"123abc"。

(3) 使用三引号(三对单引号或者三对双引号)包含字符，例如，'''abc'''。

```
"""
123
Abc
"""
```

三引号能包含多行字符串，其中可以包含换行符、制表符等特殊字符，进行格式化输出，代码如例 3-1 所示。

【例 3-1】 字符串的创建代码示例。

```
str1 = 'life'                          # 单引号
str2 = "你好,Python"                    # 双引号
print(str1)
print(str2)

str3 = '''热爱可抵岁月漫长,             # 三单引号,包含换行符\n
温柔可挡艰难时光.'''

str4 = """                             # 三双引号,包含换行符\n 和制表符\t
        勤学
        汪洙
    学向勤中得,萤窗万卷书。
    三冬今足用,谁笑腹空虚。
"""
print(str3)
print(str4)
```

输出结果如下：

```
life
你好,Python
热爱可抵岁月漫长,
温柔可挡艰难时光.

        勤学
        汪洙
    学向勤中得,萤窗万卷书。
    三冬今足用,谁笑腹空虚。
```

3.1.2 转义字符

程序执行时是严格按照语法规则进行的，所以有时候并没有大家所期望的那么灵活。

如果用单引号和双引号来创建字符串，而字符串的内容中又包含了单引号或双引号时，此时不进行特殊处理，会导致程序出现错误。

而解决方法有两种：使用不同的引号将字符串括起来和使用转义字符对引号进行转义。

1. 使用不同的引号将字符串括起来

假如字符串内容中包含了单引号，则可以使用双引号将字符串括起来。例如：

```
str3 = 'I'm a coder'
```

由于字符串中的字符包含了单引号，此时 Python 会将字符中的单引号与第一个单引号配对，这样就会把 'I' 当成字符串，而后面的 m a coder' 就变成了多余的内容，从而导致语法错误。为了避免这种问题，可以将上面代码改为如下形式：

```
str3 = "I'm a coder"
```

上面代码使用双引号将字符括起来，此时 Python 就会把字符中的单引号当成字符串内容，而不是和字符的引号配对。假如字符串中的字符包含双引号，则可使用单引号将字符括起来，例如：

```
str4 = '"Spring is here, let us jam!", said woodchuck.'
```

2. 使用转义字符对引号进行转义

Python 允许使用反斜杠符(\)将字符串中的特殊字符进行转义。若字符串既包含单引号，又包含双引号，此时必须使用转义字符，例如：

```
str5 = '"we are scared, Let\'s hide in the shade", says the bird'
```

Python 中的转义符号如表 3-1 所示。

表 3-1 转义字符

转义字符	描　述	转义字符	描　述
\(在行尾时)	续行符	\n	换行
\\	反斜杠符	\v	纵向制表符
\'	单引号	\t	横向制表符
\"	双引号	\r	回车
\a	响铃	\f	换页
\b	退格(Backspace)	\oyy	八进制数，y 代表 0～7 字符，例如，\o12 代表换行
\e	转义	\xyy	十六进制数，\x 开头，y 代表 0～9，a～f(A～F)字符，例如，\x0a 代表换行
\000	空	\other	其他的字符以普通格式输出

转义字符中\oyy 和\xyy 会输出对应的字符，代码如例 3-2 所示。

【例 3-2】 转义字符代码示例。

```
string = "\101, \x42, \71, \x63"
print(string)
```

输出结果如下：

```
A, B, 9, c
```

要计算这两种转义字符对应的结果，首先将对应八进制或十六进制转换为十进制的数字，此时十进制数字编码值对应的字符便是最后输出的结果，例如，\101，\x42，\71，\x63 分别对应的十进制数是 65，66，57，99，这四个数字对应的字符分别是 A，B，9，c。

多学一招：常用编码值如下。

65～90 对应大写字母 A～Z。

97～122 对应小写字母 a～z。

48～57 对应数字字符 0～9。

32 对应空格字符。

视频讲解

3.2 字符串格式化

3.2.1 %格式控制符

在第 2 章中介绍过 print()函数的用法，这只是最简单、最初级的形式，print()函数还有很多高级的用法，例如，想要输出可读性更好的字符串，可以运用 print()函数进行格式化输出，这就是本节要讲解的内容。

print()函数使用以%开头的格式控制符对各种类型的数据进行格式化输出，具体如表 3-2 所示。

表 3-2 格式控制符

格式控制符	解　　释	格式控制符	解　　释
%d、%i	格式化为带符号的十进制数	%g	智能选择使用%f 或%e 格式
%o	格式化为无符号的八进制数	%G	智能选择使用%F 或%E 格式
%x、%X	格式化为无符号的十六进制数	%c	格式化字符及其 ASCII 码
%e	格式化为科学记数法表示的浮点数（e 小写）	%r	使用 repr() 函数将表达式转换为字符串
%E	格式化为科学记数法表示的浮点数（E 大写）	%s	使用 str() 函数将表达式转换为字符串
%f、%F	格式化为十进制浮点数		

格式控制符相当于一个占位符，它会被后面表达式（变量、常量、数字、字符串、加减乘除等各种形式）的值代替。代码如例 3-3 所示。

【例 3-3】 单个%格式控制符的代码示例。

```
age = 8
print("小明已经%d 岁了!" % age)
```

输出结果如下：

```
小明已经 8 岁了!
```

在 print()函数中,由引号包含的是格式化字符串,它相当于一个字符串模板,可以放置一些格式控制符(占位符)。本例的格式化字符串中包含一个%d 格式控制符,它最终会被后面的变量 age 的值所替代。

age 前面的%是一个分隔符,它前面是格式化字符串,后面是要输出的表达式。

当然,格式化字符串中也可以包含多个格式控制符,同时也要提供多个表达式,用以替换对应的格式控制符;多个表达式必须使用小括号包含起来,代码如例 3-4 所示。

【例 3-4】 多个%格式控制符的代码示例。

```
name = "小明"
age = 8
print("%s 已经%d 岁了。" % (name, age))
```

输出结果如下:

```
小明已经 8 岁了。
```

另外,在使用%格式控制符时,要遵循以下原则。

(1) 数量一致。后面表达式要与前面格式控制符数量保持一致。

(2) 类型一致。后面表达式对应的数据类型要与前面格式控制符类型保持一致。

(3) 顺序一致。后面表达式会依次替换前面的格式控制符,所以顺序要保持一致。

如果数量或者类型不一致,可能会导致程序出错,如果顺序不一致,可能会导致输出的数据顺序出现错误或程序出错,代码如例 3-5 所示。

【例 3-5】 多个%格式控制符的代码示例。

```
# 数量不一致
name = "小明"
age = 8
print("%s 已经%d 岁了。" % (name))

# 类型不一致
name = "小明"
age = 8
print("%d 已经%d 岁了。" % (name, age))

# 顺序不一致
name = "小明"
age = 8
print("%s 已经%s 岁了。" % (age, name))
```

输出结果如下:

```
# 数量不一致
# 报错,提供数据不足
TypeError: not enough arguments for format string

# 类型不一致
# 报错,%d 格式化整数,而 name 是字符串类型
TypeError: %d format: a number is required, not str
```

```
# 顺序不一致
8 已经小明岁了。
```

%格式控制符还可以进行更加细致的格式修饰，%格式控制符的修饰符如表 3-3 所示。

表 3-3　%格式控制符的修饰符

符　号	功　能	符　号	功　能
－	左对齐	0	显示的数字前面填充'0'而不是默认的空格
＋	在正数前面显示加号（＋）	%	'%%'输出一个单一的'%'
<sp>	在正数前面显示空格	(var)	映射变量（字典参数）
#	在八进制数前面显示零（'0'），在十六进制前面显示'0x'或者'0X'（取决于用的是'x'还是'X'）	m.n	m 是显示的最小总宽度，n 是小数点后的位数（如果可用的话）

众多修饰符当中，使用最为频繁的是 m.n 修饰符，使用的代码如例 3-6 所示。

【例 3-6】 %格式控制符的修饰符使用的代码示例。

```
x = 23.768
print("*%f*" % x)          # 默认保留 6 位小数
print("*%10f*" % x)        # 输出数据时占宽度为 10 的位置，数据宽度不够，则左侧补空格
print("*%.5f*" % x)        # 保留 5 位小数
print("*%10.5f*" % x)      # 整个数据宽度为 10，小数位保留 5 位
print("*%3.2f*" % x)       # 输出宽度比 3 大，所以按实际输出，小数位数保留 2 位
```

输出结果如下：

```
*23.768000*
* 23.768000*
*23.76800*
* 23.76800*
*23.77*
```

多学一招：对于 m 修饰符，有两条使用规则。当指定宽度 m 大于实际宽度，左侧补零；当指定宽度 m 小于实际宽度，按实际宽度输出。

n 修饰符的使用规则：当指定小数位数 n 大于实际小数位，右侧补零；当指定小数位数 n 小于实际小数位，四舍五入保留 n 位小数。其中需要注意的是小数点和数据所带符号也是算宽度位数的。

3.2.2　format()格式化方法

Python 中字符串的格式化输出除了使用%格式控制符外，还有另外一种格式化方法，即 format()格式化方法，它增强了字符串格式化的功能。

基本语法是通过{}和 format()方法来代替%格式控制符。format()方法不限参数个数，位置可以不按顺序排列，代码如例 3-7 所示。

【例 3-7】 format()方法格式化控制的代码示例。

```
name = "小明"
age = 8
print("{}已经{}岁了。".format(name, age))
print("{1}/{0}={2}".format(3, 9, 9/3)) # 通过索引值0,1,2,…可以改变输出顺序
```

输出结果如下：

```
小明已经8岁了。
9/3=3.0
```

相较于%格式控制符，format()格式化方法有以下三个优点。

(1) 格式化时不用关心数据类型的问题，format()格式化方法会自动转换，而用%格式控制符时，需要指定数据类型，例如，%s用来格式化字符串类型，%d用来格式化整型。

(2) 单个参数可以多次输出，参数顺序可以不同。

(3) 填充方式灵活，对齐方式强大。

输出数据时，可以调用format()函数，并结合类型说明符对各种类型的数据进行格式化输出，具体如表3-4所示。

表3-4 format()格式化方法类型说明符

类型说明符	解 释	类型说明符	解 释
{:b}	输出整数的二进制形式	{:X}	输出整数的大写十六进制形式
{:c}	输出整数对应的Unicode字符形式	{:e}	输出浮点数对应的小写字母e的指数形式
{:d}	输出整数的十进制形式	{:E}	输出浮点数对应的大写字母E的指数形式
{:o}	输出整数的八进制形式	{:f}	输出浮点数的标准浮点形式
{:x}	输出整数的小写十六进制形式	{:%}	输出浮点数的百分形式

用format()格式化方法进行格式控制时，对应的修饰符与%格式控制符的修饰符大致相同，但是更丰富一些，使用的顺序如表3-5所示。

表3-5 修饰符顺序

1	2	3	4	5	6	7
":"	<填充符号>	<对齐符号>	<指定宽度>	,	<指定精度>	<类型说明符>
引导符号，在进行格式修饰时必须写	写0时可以在空格处补零；与对齐符号结合使用时可以填充符号和字母等	<：居左 ^：居中 >：居右	指定输出宽度，与%修饰符规则一致	数字的千位分隔符，适用于整数和浮点数	指定小数位数，与%修饰符规则一致	整数类型b，c，d，o，x，X，浮点数类型e，E，f，%在进行格式修饰时必须写

format()方法进行格式修饰时，修饰方式比%的格式修饰方式更加简单多样，代码如例3-8所示。

【例3-8】 format()方法进行格式修饰的代码示例。

```
x = 123.126

print("*{:^11.3f}*".format(x))    # 数据居中显示
print("*{:A^11.3f}*".format(x))   # 数据居中显示,宽度不够的位置填充大写字母 A
print("*{:011.3f}*".format(x))    # 数据左侧空格部分填充数字 0

print("*{:3.6f}*".format(x))      # 实际宽度大于 3,按实际宽度输出,小数位不够右侧补零
print("*{:011.2f}*".format(x))    # 实际宽度小于 11,左侧补零
print("*{:12.4e}*".format(x))     # 科学计数法的类型说明符是 e
```

输出结果如下：

```
* 123.126 *
*AA123.126AA*
*0000123.126*
*123.126000*
*00000123.13*
* 1.2313e+02*
```

format()格式化方法更加简单、方便，需要进行格式控制时，可以多使用 format()方法。另外，在 Python 3.6 之后，出现了 f 格式控制标识符，其中的使用规则基本与 format()方法一致，仅仅格式不同。例如，将 format()方法的程序示例进行修改，格式化效果一样，但是程序更加简化，代码如例 3-9 所示。

【例 3-9】 使用 f 格式控制标识符进行格式控制的代码示例。

```
x = 123.126
# 字符串前面加上格式标识符 f,花括号内进行格式控制时,冒号前面写变量名,冒号后面进行对应
# 格式修饰
print(f"*{x:^11.3f}")
print(f"*{x:A^11.3f}*")
print(f"*{x:011.3f}*")
print(f"*{x:3.6f}*")
print(f"*{x:011.2f}*")
print(f"*{x:12.4e}*")
```

输出结果如下：

```
* 123.126 *
*AA123.126AA*
*0000123.126*
*123.126000*
*00000123.13*
* 1.2313e+02*
```

在进行字符串格式化的学习过程中，需要循序渐进，多进行练习，才不会经常犯错。可以先使用格式比较清晰固定的 format()方法，然后在修饰符掌握比较熟练之后，慢慢过渡到使用更加简洁的 f 格式控制标识符进行格式控制。

3.3 字符串的处理

3.3.1 字符串基本操作

1. 字符串的存储和访问

Python 不支持单字符类型，单字符在 Python 中也是作为字符串使用。Python 中字符串以索引的方式存储，如果要访问字符串中的某个字符，则需要使用下标来实现。例如，字符串 name = "Python"，在内存中的存储方式如图 3-1 所示。

字符串中的每个字符都对应着两套编号：正索引和负索引。

0	1	2	3	4	5
P	y	t	h	o	n
-6	-5	-4	-3	-2	-1

图 3-1 字符串 name="Python" 在内存中的存储方式

正索引：从左到右从 0 开始，并且依次递增 1，这个编号就是下标。如果要访问字符串中的某个字符，则可以使用下标获取。例如，访问下标为 2 的字符 t，可以用 name[2]来访问。

负索引：从右往左从－1 开始，并且依次递减 1。name 变量的负索引从右往左即－1 到－6。例如，访问字符 t，还可以用 name[－4]来访问。

2. 字符串的切片操作

如果想获取字符串的某一部分内容，可以通过切片的方式来进行截取。Python 中字符串切片语法如下：

```
string[start:end:step]
```

start：切片起始位置的索引，可以用正、负索引值表示。

end：切片终止位置的索引，可以用正、负索引值表示。

step：步长，表示切片索引的增、减值，默认为 1，切片时前一个字符切完直接切片后一个字符，直到切片到终止位置的字符为止。可取正整数 1、2、3…，也可是负整数－1、－2、－3…，切片间隔数分别为 0、1、2…以此类推。

切片时需要注意的四条规则具体如下。

(1) 范围。切片时包含起始位置字符，不包含终止位置字符，例如，name[0:3]的切片结果是 Pyt，不包含 h。

(2) 省略。省略起始位置的索引，默认从第一个字符开始切片，例如，name[:3] 的切片结果仍是 Pyt。省略终止位置的索引，默认从起始位置切完为止，例如，name[1:] 的切片结果是 ython。

(3) 越界。起始位置的索引越界，默认返回空字符串，例如，name[100:6]返回空字符串。终止位置的索引越界，默认从起始位置开始切完为止，例如，name[1:100] 的切片结果是 ython。

(4) 步长。省略不写时默认值为 1，理解步长要从两个方面入手。

① 步长的正负：决定切片的方向。步长为正数，切片从左往右切片，正方向切片；步长为负数，切片从右往左，负方向切片。例如，name[1:3:1]的切片结果是 yt，而 name[3:1:－1]

的结果是 ht。

② 步长的绝对值：决定切片的间隔数。绝对值为 1 时，切片间隔数为 0，也就是一个挨着一个切片，绝对值为 2 时，切片间隔数为 1，也就是间隔一个字符再切，后面以此类推。例如，name[0:5:2]的结果是：Pto，而 name[0:5:3]的结果是：Ph。

字符串的切片操作规则较为复杂，在进行切片操作时，需要注意使用规范，代码如例 3-10 所示。

【例 3-10】 字符串切片操作的代码示例。

```
String = 'we love python'
print(String[0:4:1])
print(String[:3])
print(String[3:])
print(String[-7:-3:1])
print(String[-2:-5:-1])
print(String[6:3:-1])
print(String[::-1])
```

输出结果如下：

```
# 为方便查看程序结果,此处用下画线对空格进行了标识
we_l
we_
love_python
_pyt
oht
evo
nohtyp_evol_ew
```

切片的方式灵活多变，后续的列表和元组也会有相同的切片操作，规则跟字符串的切片操作是一样的，需要多理解切片时需要注意的四条规则，进行多次巩固练习，才能熟练地掌握切片操作，在实际开发过程中解决问题。

3.3.2 字符串运算符

在 Python 开发过程中，经常需要对字符串进行一些基本处理，例如，拼接字符串、重复输出字符串、截取字符串等。可以使用运算符对字符串进行拼接（连接）、重复等操作。字符串常用运算符如表 3-6 所示，表中实例 a="Hello"，b="Python"。

表 3-6 字符串常用运算符

操 作 符	描 述	实 例
+	字符串连接	>>> a+b 'HelloPython'
*	重复输出字符串 *n* 次	>>> a * 2 'HelloHello'
[]	通过索引获取字符串中字符	>>> a[1] 'e'

续表

操 作 符	描 述	实 例
[:]	截取字符串中的一部分	>>> a[1:4] 'ell'
in	成员运算符：如果主字符串中包含给定的子字符串则返回 True，否则返回 False	>>>"H" in a True
not in	成员运算符：如果主字符串中不包含给定的子字符串则返回 True，否则返回 False	>>>"M" not in a True
r/R	还原符：字符串包含的所有字符都是直接按照字面的意思来使用，不会进行转义	>>> r'C:\Windows\notepad.exe' C:\Windows\notepad.exe 不加 r 字符串会换行

3.3.3 字符串处理方法

在了解字符串的基本操作后，本节将介绍 Python 字符串类型常用的处理方法。在 Python 开发过程中，对字符串处理的需求多样，所以集成了很多字符串处理相关的函数和方法，如查找子字符串、字符串替换、字符串拆分等，这些操作无须开发者自己设计实现，只需调用相应的字符串方法即可。

1. 查找子字符串索引值：find()和 index()方法

find()方法用于检索字符串中是否包含目标字符串，如果包含，则返回第一次出现该字符串的索引；反之，则返回－1。

index()方法也是用于检索字符串中是否包含目标字符串，如果包含，则返回第一次出现该字符串的索引；反之，则报错。语法格式如下：

```
str.find(sub, start=None, end=None)
str.index(sub, start=None, end=None)
```

参数说明如下。

str：表示原字符串。

sub：表示待检索的子字符串。

start 和 end：可以设置子字符串查找的范围，如不设置默认范围则是整个字符串。使用 find()方法查找子字符串索引值的代码如例 3-11 所示。

【例 3-11】 使用 find()方法查找子字符串索引值的代码示例。

```
>>> study_str = "good good study, day day up"
>>> study_str.find("good")
0                                  # 返回第一次出现 good 时，g 在主字符串中的索引值
>>> study_str.find("Good")
-1                                 # 主字符串中没有 Good 子串，所以返回－1
>>> study_str.find("good", 5, 14)
5                                  # 在索引值，5 到 14 的字符串区间里面，也就是"good study"中
                                   # 检索 good 子字符串，返回对应索引值
```

同 find()方法类似，index()方法也可以用于检索字符串中是否包含指定的字符串，且对应的参数作用是一样的，不同之处在于，当指定的字符串不存在时，find()方法返回－1，

而 index()方法会出现 ValueError 报错。使用 index()方法查找子字符串索引值的代码如例 3-12 所示。

【例 3-12】 使用 index()方法查找子字符串索引值的代码示例。

```
>>> study_str = "good good study,day day up"
>>> study_str.index("day")
16
>>> study_str.index("Day")
Traceback(most recent call last):
    File "<stdin>", line 1, in <module>
ValueError: substring not found                    # 报错,子字符串没有找到
```

2. 计算子字符串出现次数：count()方法

count()方法用于检索指定字符串在主字符串中出现的次数，如果检索的字符串不存在，则返回 0，否则返回出现的次数。count()方法的语法格式如下：

```
str.count(sub, start=None, end=None)
```

各参数的具体含义如下。

str：表示原字符串。

sub：表示要检索的字符串。

start：指定检索的起始位置，也就是从什么位置开始检索。如果不指定，默认从头开始检索。

end：指定检索的终止位置，如果不指定，则表示一直检索到结尾。

使用 count()方法计算子字符串出现次数的代码如例 3-13 所示。

【例 3-13】 使用 count()方法计算子字符串出现次数的代码示例。

```
>>> study_str = "good good study,day day up"
>>> study_str.count("d")                    # 可以计算只有单个字符的字符串出现次数
5
>>> study_str.count("od")                   # 也可以计算包含多个字符的字符串出现次数
2
>>> study_str.count("Day")                  # 如果查找的字符串不存在,则返回 0
0
```

3. 字母大小写变换：lower()、upper()、title()、swapcase()方法

lower()方法用于将字符串中的所有大写字母转换为小写字母，转换完成后，该方法会返回新得到的转换为小写的字符串。如果字符串中原本就都是小写字母，则该方法会返回原字符串。

upper()方法的功能和 lower()方法恰好相反，它用于将字符串中的所有小写字母转换为大写字母，即如果转换成功，则返回小写变大写之后的新字符串；如果原字符串字母都是大写，则返回字符串与原字符串一样。

title()方法用于将字符串中每个英文单词的首字母转为大写，其他字母全部转为小写，转换完成后，此方法会返回转换得到的字符串。如果字符串中没有需要被转换的字符，此方法会将字符串原封不动地返回。

swapcase()方法是将字符串中大写字母转换为小写字母,同时把小写字母转换为大写字母。

语法规则如下:

```
str.lower()
str.upper()
str.title()
str.swapcase()
```

str 表示原字符串,四种方法不需要进行参数传递。

代码如例 3-14 所示。

【例 3-14】 使用 lower()、upper()、title()和 swapcase()方法对字符串中字母大小进行变换的代码示例。

```
>>> study_str = "Good Good Study,Day Day Up"
>>> study_str.lower()
'good good study,day day up'

>>> study_str.upper()
'GOOD GOOD STUDY,DAY DAY UP'

>>> study_str.title()
'Good Good Study,Day Day Up'

>>> study_str.swapcase()
' gOOD gOOD sTUDY,dAY dAY uP '
```

注意,以上四种方法相当于创建了一个原字符串的副本,将转换后的新字符串返回,而不会修改原字符串。

4. 字符串替换:replace()方法

replace()方法把字符串中的 old(旧字符串)替换成 new(新字符串),如果指定第三个参数 max,则替换不超过 max 次。返回的是替换后的新字符串,原字符串不会改变。replace()方法语法规则如下:

```
str.replace(old, new[, max])
```

此方法中部分参数的含义如下。

old:将被替换的子字符串。

new:新字符串,用于替换 old 子字符串。

max:可选字符串,替换不超过 max 次。

replace()方法类似于 word 文档中的查找和替换功能,可以将需要替换的文本一次性替换,代码如例 3-15 所示。

【例 3-15】 使用 replace()方法进行字符串替换的代码示例。

```
>>> study_str = "玉穷千里目,更上一层楼"
>>> study_str.replace("玉", "欲")
'欲穷千里目,更上一层楼'

>>> study_str = "玉穷千里目,更上一层楼..."
```

```
>>> study_str.replace(".", "!")              # 替换所有"."号
'欲穷千里目,更上一层楼!!!'

>>> study_str.replace(".", "!", 2)           # 替换 2 个"."号
'玉穷千里目,更上一层楼!!.'
```

5. 字符串拆分：split()方法

split()方法可以将一个字符串按照指定的分隔符切分成多个子字符串，这些子字符串会被保存到列表中(不包含分隔符)，作为返回值反馈回来。该方法的基本语法格式如下：

```
str.split(sep=None, maxsplit)
```

此方法中各参数的含义如下。

str：表示要进行分割的字符串。

sep：用于指定分隔符，可以包含多个字符。此参数默认为 None，表示所有空字符，包括空格、换行符(\n)、制表符(\t)等。

maxsplit：可选参数，用于指定分割的次数，最后列表中子字符串的个数最多为maxsplit+1。如果不指定或指定为 −1，则表示分割次数没有限制。

使用 split()方法进行字符串替换的代码如例 3-16 所示。

【例 3-16】 使用 split()方法进行字符串转换的代码示例。

```
>>> study_str = "2020 12 24"
>>> study_str.split()
['2000', '12', '24']

>>> study_str = "2020-12-24"
>>> study_str.split("-")
['2000', '12', '24']
```

6. 去除多余字符：strip()、lstrip()、rstrip()方法

用户输入数据时，可能会无意中输入多余的空格，或在一些场景中，字符串前后不允许出现空格和特殊字符，此时需要去除字符串中的空格和特殊字符。这里的特殊字符，指的是制表符(\t)、回车符(\r)、换行符(\n)等。

Python 中，字符串变量提供了 3 种方法来去除字符串中多余的空格和特殊字符，它们分别如下。

strip()方法：去除字符串前后(左右两侧)的空格或特殊字符，返回去除之后生成的新字符串，原字符串不变。

lstrip()方法：去除字符串前面(左侧)的空格或特殊字符。返回去除之后生成的新字符串，原字符串不变。

rstrip()方法：去除字符串后面(右侧)的空格或特殊字符。返回去除之后生成的新字符串，原字符串不变。

三种方法的基本语法格式如下：

```
str.strip([chars])
str.lstrip([chars])
str.rstrip([chars])
```

参数说明如下。

str：表示要进行字符去除的字符串。

char：去除字符串头、尾指定的字符序列。

使用 strip()、lstrip()和 rstrip()方法去除多余字符的代码如例 3-17 所示。

【例 3-17】 使用 strip()、lstrip()和 rstrip()方法去除多余字符的代码示例。

```
>>> study_str = " hahaha "
>>> study_str.strip()
'hahaha'

>>> study_str = "###hahaha###"
>>> study_str.strip("#")
'hahaha'
>>> study_str.lstrip("#")
'hahaha###'
>>> study_str.rstrip("#")
'###hahaha'
```

注意，Python 的 str 是不可变的(不可变的意思是指，字符串一旦形成，它所包含的字符序列不能发生任何改变)，因此这三种方法相当于只是返回字符串前面或后面字符被去除之后的副本，并不会改变字符串本身。

常用的字符串处理方法如表 3-7 所示。

表 3-7 常用的字符串处理方法

方法名称	方法说明
S.split(sep,maxsplit)	返回字符串中的字符列表，使用 sep 作为分隔符切分字符串，至多拆分 maxsplit 次
sep.join(S)	连接字符串数组。将字符串、元组、列表中的元素以指定的分隔符连接生成一个新字符串
S.isalnum()	检验字符串是否为空。如果字符串至少有一个字符，则返回 True，否则返回 False
S.replace(old,new,count)	返回字符串，其中所有的子字符串 old 用 new 替换。如果指定了可选参数 count，则只有前面 count 个子字符串 old 被替换
S.strip([char])/S.lstrip([char])/S.rstrip([char])	去除字符串的两侧/左侧/右侧空格
S.upper()/S.lower()	小写转换为大写/大写转换为小写
S.swapcase()	字符串中所有大写转为小写，所有小写转为大写
S.capitalize()	把字符串的第一个字母转为大写
S.title()	把所有单词的第一个字母转为大写
S.count(sub,start,end)	查询字符串中子字符串 sub 出现的次数，可以规定查询起、止位置
S.find(sub,start,end) S.index(sub,start,end)	查询字符串 S 中子字符串 sub 的索引值，可以规定查询起、止位置
len(S)	计算字符串 S 中的字符个数

本章小结

字符串是 Python 中最常用的数据类型。本章主要介绍了字符串类型变量的创建、字符串格式化方法(包括%格式控制符、format()格式化方法和 f 格式控制标识符)、字符串的基本操作、字符串的处理函数和方法。