第5章 字 符 串

在日常的编码中，很多时候需要对数据进行处理，无论数据是数组、列表、字典，最终都离不开对字符串的处理。字符串是字符的集合，用于存储和表示基本的文本信息，是编程语言中表示文本的数据类型。在 Python 语言中，只有用于保存字符串的 String 类型，而没有用于保存单个字符的数据类型。单个字符在 Python 中也作为一个字符串来使用。字符串是 Python 中最常用的数据类型。本章学习字符串的相关内容，包括字符串的创建、索引、拼接等操作。

5.1 字符串的表示

5.1.1 字符串的创建

若干个字符的集合就是一个字符串(String)。可以使用单引号(')、双引号(")或三引号(即三个连续的单引号'''或双引号""")来创建字符串。对于不包含任何字符的字符串，称为空字符串。创建字符串很简单，只要为变量分配一个值即可。字符串创建示例如文件 5.1 所示。

【文件 5.1】 String.py

```
str1='Hello World! '          #使用一对单引号创建字符串并赋给变量 str1
str2="你好"                    #使用一对双引号创建字符串并赋给变量 str2
str3='''我喜欢 Python'''       #使用一对三引号创建字符串并赋给变量 str3
print(str1)
print(str2)
print(str3)                   #输出三个字符串
```

程序运行的结果如下。

```
Hello World!
你好
我喜欢 Python
```

5.1.2 字符串的转义

在字符串中，反斜杠“\”具有转义的作用。

（1）将“\”单独放在较长字符串某一行的行尾，表示续行，即下一行与当前行是同一行。

（2）如果字符串中本身含有“\”，为了使字符串能正常输出，需要让反斜杠“\”失去转义功能。此时有两种方法，可以使用连续的两个反斜杠“\\”来表示字符“\”；或者在字符串的开头加上 r 前缀。具体示例如下。

```
print('This is line1.\
This is line2.')
print('This is line1.\\This is line2.')
print(r'This is line1.\This is line2.')
```

运行结果如下。

```
This is line1.This is line2.
This is line1.\This is line2.
This is line1.\This is line2.
```

（3）“\'”与“\"”分别表示输出单引号和双引号本身。

常用转义字符见表 5-1。

表 5-1 常见转义字符

转义字符	含　义	转义字符	含　义
\(在行尾时)	续行符	\n	换行
\\	反斜杠\	\r	回车
\'	单引号'	\t	水平制表符
\"	双引号"		

5.1.3 引号的区别

单引号与双引号二者在使用方法上并没有什么区别。使用单引号或双引号表示字符串时，如果字符串内容中出现引号，需要进行特殊处理，否则会出现错误。例如 'I'm a great coder! '，由于该字符串中包含单引号，此时 Python 会将字符串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 great coder! '就变成了多余的内容，从而导致语法错误。在这种情况下，有两种解决方案。

（1）对引号进行转义。在引号前面添加斜杠\就可以对引号进行转义，让 Python 把它作为普通文本对待，例如：

```
str1 = 'I\'m a great coder! '   #使用\'说明该单引号是字符串中的一个字符，不加\会报错
```

```
str2 ="He said:\"It is a book.\""        #使用\"说明该双引号是字符串中的字符
print(str1)
print(str2)
```

运行结果：

```
I'm a great coder!
He said:"It is a book."
```

(2) 使用不同的引号包围字符串。如果字符串内容中出现了单引号，可以使用双引号包围字符串。反之亦然。

```
str1 ="I'm a great coder!"               #使用双引号包围含有单引号的字符串
str2 ='He said:\"It is a book.\"'        #使用单引号包围含有双引号的字符串
```

此时输出结果与上面相同。

在 Python 中，程序的换行、缩进都有严格的语法要求。单引号与双引号中的字符串通常要求写在一行中，单引号和双引号中的字符串如果分多行写，必须在每行的结尾加上续行符“\”；如果希望一个字符串中包含多行信息，需要使用换行符“\n”。例如：

```
s1='Hello \
World! '                          #第一行以\作为结尾，说明第二行与第一行是同一条语句
s2="你好，\n 世界!"                #通过\n 进行换行
print(s1)
print(s2)
```

运行结果：

```
Hello World!
你好，
世界!
```

在字符串较长的情况下，可以使用三引号进行创建。使用三引号可以直接将字符串写成多行的形式或直接进行换行(不需要加\或\n)。长字符串中的换行、空格、缩进等空白符都会原样输出。

语法格式：

```
str='''你好!
欢迎学习 Python,
祝你学习愉快。'''                    #不需要加\n，输出时会自动换行
print(str)
```

运行结果：

```
你好!
欢迎学习 Python,
祝你学习愉快。
```

此外，在三引号包围的字符串中可以直接放置单引号或者双引号，不需要使用转义符。不会导致解析错误。例如：

```
str='''He said: ''It's your book.'''''      #字符串中的单引号与双引号都正常输出
```

程序运行的结果如下。

```
He said:''It's your book.'
```

如果在 Python 程序中出现由一对三引号包围的字符串，且没有赋值给任何变量，那么这个长字符串就不会起到任何作用，可以当作注释使用。

注意，此时 Python 解释器并不会忽略长字符串，也会按照语法解析，只是起不到实际作用而已。

5.2 字符串的索引和切片

5.2.1 字符串序号

如果需要访问字符串中的一个字符，需要知道它在字符串中的位置。可以通过正向递增序号或反向递减序号对字符进行编号。字符串序号示例如表 5-2 所示。

表 5-2 字符串序号示例

字符串	p	y	t	h	o	n
正序	0	1	2	3	4	5
倒序	−6	−5	−4	−3	−2	−1

5.2.2 字符串索引与切片

通过索引(下标)可以精确地定位到字符串中的某个元素，并使用 str[num]获取该字符，num 为该字符的序号。例如：

```
str="Python"
print(str[0])                    #输出 p
print(str[-1])                   #输出 n
```

获取字符串中的多个字符，可以使用切片的方法进行截取。语法格式为 str[start:end:step]。其中，str 是待切片的字符串；start 是要切片的第一个字符的序号(包括该字符)，如果不指定默认为从头开始；end 表示切片的最后一个字符的序号(不包括该字符)，如果不指定默认至字符串结尾；step 表示切片的步长，如果不指定默认为 1。字符串截取的示例如文件 5.2 所示。

【文件 5.2】 String_slice.py

```
str ="Python"
print(str[:])            #截取字符串中的全部字符
print(str[1::2])         #从第二个字符至结尾(包括最后一个字符)，以步长为 2 进行切片
print(str[:5:3])         #从开头至第五个字符(包括第一个，不包括第六个)，步长为 3
print(str[2:4])          #截取第三个到第四个字符，默认步长为 1
```

```
print(str[::-1])          #字符串逆序输出,步长为-1表示从后向前逐一输出
```

运行结果：

```
Python
yhn
ph
th
nohtyp
```

5.3 字符串常用方法

5.3.1 字符串检索

字符串提供了4种检索方法,分别是find()，index()，rfind()，rindex()。4种方法的语法格式及参数相同。以find()方法为例,语法格式为

```
str.find(sub[,start[,end]])
```

上述4种方法的作用都是从字符串str中检索字符串sub出现的位置。start与end参数指定了检索范围,如不指定则默认在str[:](即整个字符串范围)中进行检索。

find()方法是在指定检索范围中按照从左向右的顺序进行检索,并找到字符串sub第一次出现的位置;而rfind()方法是在指定范围内按照从右向左的顺序进行检索,并找到字符串sub第一次出现的位置。

index()方法与find()作用相同,rindex()与rfind()作用相同。区别在于,当字符串str中检索不到sub时,find()与rfind()方法返回－1,而index()与rindex()会引发异常。

find()与rfind()方法使用示例如下。

```
str='Python java Python c'
print('第一次出现 Python 的位置:',str.find('Python'))
print('最后一次出现 Python 的位置:',str.rfind('Python'))
```

输出如下。

```
第一次出现 Python 的位置: 0
最后一次出现 Python 的位置: 12
```

此外,在字符串中,可以通过str.startswith('sub')和str.endswith('sub')两个函数来判断字符串是否以指定的字符串开始或结束,并根据结果输出True或False。其中,sub为指定的字符串。示例如下。

```
str='Hello world'
print(str.startswith('He'))          #判断字符串 str 是否以 He 开头
print(str.endswith('d'))             #判断字符串 str 是否以 d 结尾
print(str.endswith('s'))             #判断字符串 str 是否以 s 结尾
```

输出如下。

```
True
True
False
```

5.3.2 字符串的替换

使用字符串中的 replace()方法可以将字符串中的指定子字符串替换为其他内容，其语法格式为 str.replace(old,new[,max])。其中，str 是待操作字符串；old 和 new 分别指要替换的子串与新字符串；max 是最多替换的子串数量，如不指定该参数，则将所有满足条件的子串替换掉。replace()方法返回替换之后的字符串。

```
str='abcabcabc'
str1=str.replace('a','x',2)          #将字符串中的 a 替换为 x,且仅替换两个
str2=str.replace('bc','y')           #将字符串中的 bc 替换为 y,全部替换
print(str1)                          #输出 xbcxbcabc
print(str2)                          #输出 ayayay
```

5.3.3 字符串切割

字符串切割有两种常用方法，分别是 split()方法与 splitlines()方法。

(1) split()方法。使用 split()函数可以按照指定的分隔符对字符串进行切割，返回由切割结果组成的列表。该方法语法格式为

```
str.split(sep,maxsplit)
```

其中，str 是待切割的字符串；sep 表示指定的分隔符，可以由一到多个字符组成，不指定分隔符则默认按照空白符(空格、换行、制表符)进行切割；maxsplit 决定最大切割次数，如果指定 maxsplit 的值，则最多可以得到 maxsplit+1 个切割结果，不指定则默认 maxsplit 值为-1，表示对切割次数不做限制。

split()函数使用示例如文件 5.3 所示。

【文件 5.3】 String_split.py

```
str1='It is a book! '
str2='Python#C++#Java#PHP'
ls1=str1.split()       #默认按照空格对 str1 进行切割,且分割次数无限制
ls2=str2.split('#')    #指定#作为分隔符进行切割,分割次数无限制
ls3=str2.split('#',2)  #指定#作为分隔符进行切割,分割次数为 2 次,可以得到 3 个分割结果
print('ls1:',ls1)
print('ls2:',ls2)
print('ls3:',ls3)      #切割结果列表分别保存在 ls1,ls2,ls3 中,进行输出
```

输出结果如下。

```
ls1: ['It', 'is', 'a', 'book! ']
ls2: ['Python', 'C++', 'Java', 'PHP']
ls3: ['Python', 'C++', 'Java#PHP']
```

(2) splitlines()方法。splitlines()函数固定以行结束符('\n', '\r', '\r\n')作为分隔符对字符串进行切割,即按行对字符串进行切割,并返回由切割结果组成的列表。该方法语法格式为

```
str.splitlines(keepends)
```

其中,str 是待切割字符串;keepends 表示切割结果是否需要保留最后的行结束符,如果该参数值为 True,则保留行结束符,否则不保留。如果不指定该参数值,则默认为 False,即不保留行结束符。

splitlines()函数使用示例如下。

```
str='Python\nC++\r\nJava\r'        #字符串 str 含有三个行结束符
ls1=str.splitlines()               #默认不保留行结束符
ls2=str.splitlines(True)
print('ls1:',ls1)
print('ls2:',ls2)
```

输出结果如下。

```
ls1: ['Python', 'C++', 'Java']
ls2: ['Python\n', 'C++\r\n', 'Java\r']
```

5.3.4 字符串的连接

作为一种序列数据,可以直接使用加号(+)连接两个字符串。此外,还可以使用字符串中的 join()方法将字符串或序列中的元素以指定的字符连接成一个新的字符串。join()方法的语法格式为 str.join(seq)。其中,seq 是待连接的字符串或序列,str 是连接符。字符串连接示例如文件 5.4 所示。

【文件 5.4】 String_join.py

```
print('Py'+'thon')                 #直接使用加号连接两个字符串
s1='Python'
print('*'.join(s1))                #使用 * 作为连接符,对 s1 中的字符进行连接
s2=['Hello','world']               #构建字符串序列
print(' '.join(s2))                #使用空格作为连接符,对序列 s2 中的元素进行连接
```

输出结果依次如下。

```
Python
p*y*t*h*o*n
Hello world
```

5.3.5 去除字符串空格

去除字符串头部或尾部空格,可以使用字符串中的 strip()、lstrip()以及 rstrip()方法去除头部和尾部空格。语法格式如下。

```
str.strip()                        #去除字符串 str 中头部与尾部的空格
```

```
str.lstrip()                    #去除字符串 str 中头部的空格
str.rstrip()                    #去除字符串 str 中尾部的空格
```

去除字符串所有空格，主要使用以下两种方法。

(1) replace()方法。使用 str.replace('',")可以去除 str 中的所有空格，表示使用空字符来代替字符串中的所有字符。

(2) split()方法＋join()方法。先通过 split()按照空格对字符串进行切割，返回由切割结果组成的列表，再使用空字符将序列中的元素连接成一个新的字符串。

两种方法示例如下。

```
#方法 1
s='I likePython'
print(s.replace(' ',''))      #使用 replace()方法去除空格
#方法 2
s1=s.split()                  #先通过 split()返回切割列表['I', 'like', 'Python']
s2=''.join(s1)                #连接列表元素
print(s2)
#两种方法输出结果均为 IlikePython
```

5.3.6 字符串比较

两个字符串的比较按照从左至右的顺序逐个字符比较，如果对应两个字符相同，则继续比较下一个字符。如果找到了两个不同的对应字符，则具有较大 ASCII 码的字符对应的字符串具有更大的值，此时无须继续比较剩余字符。

如果对应字符都相同且两个字符串长度相同，则这两个字符串相等；如果对应字符都相同但两个字符串长度不同，则较长的字符串具有更大的值。字符串比较示例如文件 5.5 所示。

【文件 5.5】 String_comparison.py

```
str1='Python'
str2='C++'
str3='Python3.7'
str4='Python'                 #P 的 ASCII 码是 80,C 是 67
print(str1>str2)   #比较字符串。判断 str1 是否大于 str2,是则输出 True,不是输出 False
print(str1>=str3)             #判断 str1 是否大于或等于 str3
print(str1==str4)             #判断 str1 是否等于 str4
```

比较结果如下。

```
True
False
True
```

Python 中的字符串比较是区分大小写的。如果想以不区分大小写的方式进行字符串比较，可以使用 str.lower() 方法将字符串中的所有字符转换为小写，然后继续进行比较。

此外，字符串中有多种用于大小写转换的相关方法，例如：

```
str.capitalize()                #将字符串中的首字母大写,其他小写
str.title()                     #将字符串中每个单词的首字母大写,其他小写
str.lower()                     #所有字母小写
str.upper()                     #所有字母小写
str.swapcase()                  #将小写字母变大写,大写字母变小写
```

5.4 字符串处理函数

Python 中提供了 6 种字符串处理函数。

(1) len(x)：返回字符串的长度。注意：在 Python 中标点符号以及空格同样是字符长度。例如：

```
x='第一节课上 Python.'
len(x)                          #输出长度为 12
```

(2) str(x)：可以将任意类型的 x 转换成字符串形式(增加引号)。例如：str(1.23)＝"1.23"。

(3) eval(x)：与 str(x)的作用相对。可以将字符串转变为 Python 可以运行的语句,并执行该字符串表达式,并返回该表达式的值。也可理解为去掉字符串两侧的引号。例如：

```
eval("2+3")                     #将字符串转变为表达式 2+3,并计算输出结果 5
eval("'py' in 'Python'")        #输出 True
a="[1,2,3,4]"
print(eval(a))                  #a 是字符串类型,输出列表型[1,2,3,4]
```

(4) hex(x)或 oct(x)：把整数 x 变成十六进制或者八进制的字符串。

```
hex(100),oct(100)
('0x64', '0o144')               #输出结果
```

(5) chr(x)：x 为 Unicode 编码,返回对应的字符。

(6) ord(x)：与 chr(x)作用相对,返回对应字符的 Unicode 码。

```
chr(37)                         #输出为:'%'
ord('a')                        #输出为:97
```

5.5 字符串操作符

5.5.1 字符串运算符

Python 提供了几种用于字符串运算的操作符,分别是＋、＊、in、not in,见表 5-3。

表 5-3 字符串操作符

操作符	描　　述
+	连接两个字符串
'x' * n 或 n * 'x'	将字符串 x 重复 n 次输出
in	如果字符串中包含给定字符，返回 True
not in	如果字符串中不包含给定字符，返回 True

操作符使用方法如下。

```
print('Py'+'thon')                    #将 Py 与 thon 连接到一起
print('Py' * 2)                       #输出 2 次 Py,与 2 * 'Py'含义相同
print('Py' in 'Python')
print('x' not in 'Python')            #分别判断 Py 与 x 是否在字符串 Python 中
```

输出结果如下。

```
Python
PyPy
True
True
```

5.5.2 is 身份运算符

Python 中使用 is 操作符来比较字符串。如果两个变量具有相同的内存位置，它们的身份就被认为是相同的，并返回比较结果 True 或 False。对于字符串类型的变量，使用 is 运算符和==关系运算符效果一致。但对于列表型，==测试的是相等性，作用在于比较两个变量所指代的含义是否相同。而 is 运算符判断的是同一性，它的作用在于比较两个变量是否指向了同一个对象。is 运算符的示例如文件 5.6 所示。

【文件 5.6】 String_operator.py

```
#字符串类型
s1='hello'
s2='hello'
s1==s2        #True
s1 is s2      #True

#列表类型
x=y=[1, 2, 3]
z=[1, 2, 3]   #x 和 y 都绑定到同一个列表,而 z 被绑定在另外一个具有相同数值和顺序的列表上
x==y          #True
x==z          #True
x is y        #True
x is z        #False
#虽然两个列表的值相等,但它们是不同的变量
```