第 5 章　字符串与正则表达式

本章学习目标

- 了解字符串的概念。
- 掌握字符串的基本操作、处理函数和处理方法。
- 掌握普通字符和特殊字符正则表达式的构造和使用方法。
- 掌握 re 模块的使用方法。

5.1　字符串基本操作

字符串是 Python 中最常用的数据类型，它是连续的字符序列，一般使用单引号(' ')、双引号(" ")或三引号(''' ''')进行界定。其中，单引号和双引号中的字符序列必须在一行上，而三引号内的字符序列既可以分布在一行上，也可以分布在连续的多行上。需要注意的是，引号必须配对使用。例如，不能以双引号开始、单引号结束。字符串是一个不可变类型，即声明之后，其中的所有字符都不能再修改。

5.1.1　字符串的创建与删除

1. 字符串的创建

只要将字母、数字、汉字或其他符号用一对引号括起来，就可以获得字符串。如，' abc'、'35'、'True'、'3＋5'、"你好！"、'''Good morning'''、"xyz'wh'ef"都是合法字符串，空字符串可以表示为''、""或''''''。

使用 Python 内置函数 str()可以将列表、元组、字典、集合、数字等其他类型转换为字符串类型，示例如下：

```
>>> lst=[1,2,3,4]
>>> str(lst)
'[1, 2, 3, 4]'
>>> dct=dict(a=1,b=2,c=3)
>>> str(dct)
"{'a': 1, 'b': 2, 'c': 3}"
>>> str(1+2)
'3'
>>> str(True)
'True'
```

2. 字符串的删除

不需要的字符串对象可以使用 del 命令删除,已经删除的字符串变量不能再次访问,否则将会报错。示例如下:

```
>>> s='abc'
>>> del s
```

5.1.2 字符串的连接与扩展

虽然字符串创建后其中的元素值是不允许修改的,但是可以对字符串进行连接和扩展,从而生成新的字符串,示例如下:

```
>>> s1='hello'
>>> s2='world'
>>> s1+s2
'helloworld'
>>> 'hello'  'world'                    #字符串之间使用一个或多个空格连接
'helloworld'
>>> 'hello''world'                      #字符串之间无任何符号也可以连接
'helloworld'
>>> s1*2                                #通过重复的方式扩展字符串
'hellohello'
```

从以上代码可以看出,我们可以使“+”运算符或者空格将多个字符串连接成一个新字符串,或者将字符串放在一起,之间无任何连接符也可以连接多个字符串。可以将一个字符串中的所有元素通过“*”运算符进行重复扩展,但是原有的字符串对象本身不会发生变化。

5.1.3 字符串的长度计算

使用 Python 内置函数 len()可以统计字符串的长度。在字符串中,一个汉字、一个字母、一个数字、一个符号、一个空格都是一个字符。示例如下:

```
>>> s='学习 Python'
>>> len(s)
8
>>> s='hello world'                     #字符串里有一个空格
>>> len(s)
11
>>> s='hello\nworld'                    #\n 为转义字符,表示换行
>>> len(s)
11
>>> s='J\141v\141'                      #\141 为转义字符,用 3 位八进制表示字符'a'
>>> len(s)
4
>>> s
'Java'
```

需要注意的是,字符串中的转义字符在统计长度时只能算作一个字符。

5.1.4　字符串的索引和切片

1. 字符串的索引

字符串的索引和列表、元组元素索引一样，分为正向索引和反向索引，正向索引元素编号从 0 开始，反向索引元素编号从－1 开始。也可以使用下标索引来访问字符串中的元素。具体示例如下：

```
>>> s='学习 Python 程序设计语言'
>>> s[0]
'学'
>>> s[-1]
'言'
```

可以使用循环结构来访问字符串中的每个元素，下面的一段代码实现了遍历字符串中的每个元素。

```
s= '风景优美'
for i in s:
    print(i,end=' ')
```

运行结果为：

```
风 景 优 美
```

2. 切片操作

字符串也可以像列表、元组一样，采用切片的方式来获取指定的部分字符。下面是一些切片操作的具体示例：

```
>>> s[:8]                          #从头选取到索引位置 7 所对应的元素，步长为 1
'学习 Python'
>>> s[8:]                          #从索引位置 8 选取到字符串的尾部，步长为 1
'程序设计语言'
>>> s[2:8]                         #从索引位置 2 选取到索引位置 7 所对应的字符，步长为 1
'Python'
>>> s[:]                           #选取整个字符串
'学习 Python 程序设计语言'
```

5.1.5　成员资格判断

字符串可以采用成员运算符 in、not in 判断字符串中是否存在指定的元素。示例如下：

```
>>> s='学习 Python 程序设计语言'
>>> '程序' in s
True
>>> '计算机程序' not in s
True
>>> 'python' in s
False
```

字符串成员资格判断也可以使用字符串对象的 index()方法、rindex()方法、count()方法、find()方法,这些方法的具体使用将在 5.2 节分别介绍。

5.2 字符串常用方法

5.2.1 字符串索引

字符串索引方法有 index()和 rindex()。index()方法根据指定范围返回一个字符串在当前字符串中首次出现的位置,如果此字符串不存在,则抛出异常。rindex()方法根据指定范围返回一个字符串在当前字符串中最后一次出现的位置,如果此字符串不存在,则抛出异常。两种方法的使用格式如下:

```
strname.index(obj[,start,[stop]])
strname.rindex(obj[,start,[stop]])
```

strname 表示字符串名称,obj 表示待搜索的字符串,start、stop 分别表示搜索范围的起始位置与终止位置。如果不指定搜索范围,则默认搜索整个字符串。示例如下:

```
>>> s='Python 是程序设计语言,程序设计语言有多种'
>>> s.index('程序')
7
>>> s.rindex('程序')
14
```

5.2.2 字符统计和查询

1. 字符统计

count()方法用于统计一个字符串在当前字符串出现的次数,其一般格式为:

```
strname.count(obj)
```

obj 表示待统计出现次数的字符元素。如果该方法返回一个大于 0 的整数,则表示被统计的字符存在于字符串中。示例如下:

```
>>> s='Python 是程序设计语言,程序设计语言有多种'
>>> s.count('程序')
2
```

2. 字符查询

find()方法用于查询一个字符串在当前字符串中出现的索引位置,其一般格式为:

```
strname.find(obj[,start,[stop]])
```

obj 表示待查询的字符串。start 是查询的开始位置,stop 是查询的结束位置,如果不指定查询范围,则默认查询整个字符串。如果待查询的字符串存在于原字符中,则返回 obj 第一个字符在原字符串中的位置,如果不包含,则返回-1。示例如下:

```
>>> s='Python is a programming language.'
>>> s.find("is")
7
>>> s.find("Is")
-1
>>> s.find("is",1,6)
-1
```

5.2.3　字符串的替换

字符串的替换可以使用 replace()方法，其一般格式为：

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

它表示把字符串中的 old(旧字符串)替换成 new(新字符串)，如果指定第 3 个参数 max，则替换不超过 max 次。需要注意的是，replace()方法返回一个新的字符串，不修改原有的字符串。示例如下：

```
>>> s='Python 是程序设计语言,程序设计语言有多种'
>>> s.replace('程序设计', '编程')
'Python 是编程语言,编程语言有多种'
>>> s
'Python 是程序设计语言,程序设计语言有多种'
```

另外，字符串的替换还可以结合使用 maketrans()、translate()两种方法。

1. maketrans()方法

maketrans()方法的一般使用格式为：

```
strname.maketrans(intab, outtab)
```

maketrans()方法用于创建字符映射的转换表，第一个参数 intab 是字符串，表示需要转换的字符，第二个参数 outtab 也是字符串，表示转换的目标。两个字符串的长度必须相同，为一一对应的关系。

2. translate()方法

translate()方法的一般使用格式为：

```
strname.translate(table)
```

table 表示转换表，转换表是通过 maketrans 方法转换而来的。

translate()方法对字符串中的具体字符根据 table 转换表中的字符映射关系进行替换。两种方法的应用示例如下：

```
>>> table='' .maketrans('0123456789', '零壹贰叁肆伍陆柒捌玖')
>>> 'Tel:563270'.translate(table)
'Tel:伍陆叁贰柒零'
```

5.2.4 字符串的分隔与连接

1. 字符串的分隔

字符串的分隔可以使用 split()和 rsplit()方法,它们的一般使用格式为:

```
strname.split([separator[,maxsplit]])
strname.rsplit([separator[,maxsplit]])
```

separator 表示分隔符,若不指定分隔符,则默认空白字符为分隔符。maxsplit 表示分隔的最大次数,如果不指定分隔次数,则按照字符串中所有出现的指定分隔符进行分隔。

split()方法使用指定的字符串(默认空白字符)作为分隔符对原字符串从左向右进行分隔。rsplit()方法使用指定的字符串(默认空白字符)作为分隔符对原字符串从右向左进行分隔。这两种分隔方法都能获得一个新列表,原有的字符串不改变。示例如下:

```
>>> s='Python is a programming language.'
>>> s.split()                                  #从左向右默认以空白字符分隔
['Python', 'is', 'a', 'programming', 'language.']
>>> s.rsplit(maxsplit=2)                       #从右向左默认以空白字符分隔,分隔两次
['Python is a', 'programming', 'language.']
```

2. 字符串的连接

join()方法用于将序列中的元素以指定的字符连接生成一个新的字符串,它的一般使用格式为:

```
connector.join(sequence)
```

connector 表示字符串的连接符,sequence 表示待连接的序列对象。示例如下:

```
>>> lst=['Python', 'is', 'a', 'programming', 'language.']
>>> ' '.join(lst)                              #以空格符连接列表中的元素
'Python is a programming language.'
>>> tp=('P','y','t','h','o','n')
>>> ''.join(tp)                                #以空字符连接元组中的元素
'Python'
>>> ':'.join(map(str,range(1,10,2)))           #以冒号连接 map 对象中的元素
'1:3:5:7:9'
```

5.2.5 字符串中字母大小写转换

用于字符串中字母大小写转换的方法有 lower()、upper()、capitalize()、title()、swapcase(),这些方法在字母转换上是各不相同的,它们的具体功能如下。

- lower():把字符串中的英文字母全部转换为小写字母。
- upper():把字符串中的英文字母全部转换为大写字母。
- capitalize():把每个句子的第一个字母转换为大写字母。
- title():把每个单词的第一个字母转换为大写字母。
- swapcase():把字符串中的小写英文字母全部转换为大写字母,并把字符串中的大

写英文字母全部转换为小写字母。

示例如下：

```
>>> s='Python is a programming language.'
>>> s.lower()
'python is a programming language.'
>>> s.upper()
'PYTHON IS A PROGRAMMING LANGUAGE.'
>>> s.capitalize()
'Python is a programming language.'
>>> s.title()
'Python Is A Programming Language.'
>>> s.swapcase()
'pYTHON IS A PROGRAMMING LANGUAGE.'
```

5.2.6　字符串的对齐

用于设置字符串对齐的方法主要有 ljust()、rjust()、center()，这些方法的一般使用格式与功能分别如下。

1. 左对齐

```
strname.ljust(width[, fillchar])
```

width 表示设置对齐后字符串的总长度，如果指定的长度小于原字符串的长度，则返回原字符串。fillchar 表示填充字符，若设置对齐时字符串的总长度大于原字符串的长度，则用 fillchar 指定的符号填充，以达到规定的长度。如果不指定 fillchar，则默认以空格填充。

ljust()方法返回一个原字符串左对齐，并使用指定的符号(默认空格)填充至指定长度的新字符串。

2. 右对齐

```
strname.rjust(width[, fillchar])
```

width、fillchar 参数在 rjust()方法中的使用同 ljust()方法。

rjust()返回一个原字符串右对齐，并使用指定的符号(默认空格)填充至指定长度的新字符串。

3. 居中对齐

```
strname.center(width[, fillchar])
```

width，fillchar 参数在 center()方法中的使用同 ljust()方法。

center()方法返回一个原字符串居中对齐，并使用指定的符号(默认空格)填充至指定长度的新字符串。

3 种对齐方法的应用示例如下：

```
>>> s='Python'
>>> s.ljust(10)                          #右侧使用空格填充
'Python    '
>>> s.ljust(4)                           #若设置的长度小于字符串总长度，则返回原字符串
'Python'
```

```
>>> s.rjust(10, '%')                    #左侧使用%填充
'%%%%Python'
>>> s.center(10, '*')                   #左右两侧使用*填充
'**Python**'
```

5.2.7 字符串的测试

测试字符串中的开始或结束字符串可以使用 startswith()和 endswith()方法,它们的一般使用格式为:

```
strname.startswith(testchar)
strname.endswith(testchar)
```

testchar 表示待测试字符串,如果待测试的字符串有多个,则 testchar 用元组表示。

startswith()、endswith()分别用来测试字符串是否以指定的一个或几个字符串(放在元组中)开始和结束,返回值为 True 或 False。示例如下:

```
>>> s='Python is a programming language.'
>>> s.startswith('Python')              #测试字符串是否以'Python'开始
True
>>> s.startswith('python')
False
>>> s.endswith((';','!','.'))          #测试字符串是否以元组中的某个字符串结束
True
```

5.2.8 字符串两侧字符的删除

删除字符串两侧的字符可以使用的方法有 lstrip()、rstrip()、strip(),这些方法的具体功能各不相同,它们的一般使用格式分别如下:

```
strname.lstrip([delchar])
strname.rstrip([delchar])
strname.strip([delchar])
```

delchar 表示待删除的字符,如果不指定该参数,则默认删除空白字符。

lstrip()、rstrip()、strip()方法分别用来删除字符串左侧、右侧、两侧的空白字符或指定字符。示例如下:

```
>>> s='   %%Python is a programming language.%%   '
>>> s.lstrip()                                  #删除左侧的空白字符
'%%Python is a programming language.%%   '
>>> s.rstrip()                                  #删除右侧的空白字符
'   %%Python is a programming language.%%'
>>> s.strip()                                   #删除两侧的空白字符
'%%Python is a programming language.%%'
>>> s.strip(' %')                               #删除两侧的空白字符和%
'Python is a programming language.'
```

5.2.9　字符串中的字符判断

判断一个字符串由什么类型的字符组成，通常使用的方法有以下几种。

1. strname.isalnum()

判断字符串是否由字母或数字组成，如果字符串由字母或数字组成，则返回 True，否则返回 False。

2. strname.isalpha()

判断字符串是否全部由字母组成的，如果字符串全部由字母组成，则返回 True，否则返回 False。

3. strname.isdigit()

判断字符串是否全部由数字组成，如果字符串全部由数字组成，则返回 True，否则返回 False。

4. strname.isspace()

判断字符串是否全部由空格组成的，如果字符串全部由空格组成，则返回 True，否则返回 False。

5. strname.islower()

判断字符串是否全部是小写，如果字符串全部是小写，则返回 True，否则返回 False。

6. strname.isupper()

判断字符串是否全部是大写，如果字符串全部是大写，则返回 True，否则返回 False。

7. str.istitle()

判断字符串首字母是不是大写，如果字符串首字母是大写，则返回 True，否则返回 False。

示例如下：

```
>>> s='Python 3.9'
>>> s.isalnum()
False
>>> s.isalpha()
False
>>> s.isdigit()
False
>>> s.isspace()
False
>>> s.islower()
False
>>> s.isupper()
False
>>> s.istitle()
True
```

5.2.10　格式化字符串

利用 format()方法可以将数据按照格式化字符串所指定的格式输出。第 2 章介绍了

format()方法的简单应用,例如:

```
>>> 'My name is {0}, and I am {1}. '.format('Mary', 8)
'My name is Mary, and I am 8. '
>>> 'My name is {0}, and I am {age}. '.format('Mary', age=8)
'My name is Mary, and I am 8. '
```

本节将详细介绍格式化规则在 format()方法中的应用,其一般使用格式为:

```
'{数据编号:格式化规则}'.format(obj)
```

obj 表示按照指定的格式化规则输出的数据,可以是一个,也可以是多个数据。'{数据编号:格式化规则}'称为格式化字符串,其中的数据编号对应 format()方法中的某个数据,例如,若数据编号为 0,则对应 format()方法的第一个数据;若数据编号为 1,则对应 format()方法的第 2 个数据,以此类推。format()方法里的每个数据都可以使用多次。格式化规则给出了 format()方法中数据的显示格式。

格式化规则的书写方法如下:

```
[对齐方式][符号显示规则][#][0][填充宽度][千分位分隔符][.<小数精度>][显示类型]
```

需要注意的是,书写规则时,除非对应部分不出现,否则就必须严格按照上述顺序,不能修改。下面分别介绍格式化规则的各部分内容。

1. 对齐方式

在格式化规则中,对齐方式主要有居中、左对齐、右对齐,用符号表示分别是^、<、>。这些对齐符号前面通常带有填充字符,且填充字符只能是一个字符,若不指定填充字符,则默认用空格填充。

2. 符号显示规则

符号显示规则的取值在默认情况下是"-",即表示只有负数才显示符号,正数不显示。

符号显示规则的取值为"+"表示无论正数还是负数,都显示符号。

符号显示规则为"空格符"表示正数在符号位显示一个空格,负数显示负号。

3. # 或 0

符号显示规则后面的"#"表示如果是以二进制显示数字,则显示前导符 0b;如果以八进制显示数字,则显示前导符 0o;如果以十六进制显示数字,则显示前导符 0x。

在格式化规则中,"#"后面的 0 表示将对齐符号前面的填充字符修改为 0。

4. 填充宽度

在格式化规则中,填充宽度设置数据输出时的长度,如果填充宽度小于数据本身的长度,则按照原有的数据输出。

5. 千分位分隔符

在格式化规则中,千分位分隔符只有两种取值,即","和"_"。

6. 小数精度

小数精度是指显示的小数位数,不足的部分用 0 补齐。

7. 显示类型

在格式化规则中,最后一部分是"显示类型",表示数据的类型。例如,字符串类型用 s 表示。表 5-1 和表 5-2 给出了整数类型和小数类型的可用形式。