第 5 章

CHAPTER 5

字符文本操作

字符串是 Python 常用的数据类型之一，我们可以使用引号创建字符串。字符串支持三种不同类型(单引号、双引号、三引号)的形式，这在第 2 章中已经做了介绍。本章重点介绍字符串操作的相关知识，包括字符串查找匹配、文本处理、相似性比较等内容。

5.1 重新认识字符串

定义字符串，使用单引号、双引号或三引号没有区别，类型都是一样的，并且字符串是递归型的数据类型，即字符串的子串还是字符串类型，如以下定义所示。

```
IN [1]:s1 = '东北大学:NEU'
       s2 = "东北大学:NEU"
       print(type(s1),type(s2))
       print(type(s1[0]),type(s2[-1]))
OUT [1]:<class 'str'> <class 'str'>
       <class 'str'> <class 'str'>
```

可见，Python 字符串的类型名是 str，是含有 Unicode 字符的不可变数组，每个字符串里的子串也还是 str 类型。既然 str 数据类型是数组，那么很多和序列通用的操作在字符串上也同样可以应用，如索引、切片、长度计算 len 函数等。字符串是不可变类型，因此任何修改字符串的行为操作都会引起抛出异常，这在前面章节已经讲述，这里不赘述。需要注意的是，Python 3 版本之后都采用 Unicode 编码，不再像之前版本有各种编码的问题，这里我们首先了解一下计算机编码这方面的历史。

计算机最早由以美国为代表的英语系国家发明和主导，因此，最早的定义包含英语字符的字符编码规范，即 ASCII 码，用 8bit 位(即 1 字节)数值表示一个字符，编码值范围为[0,255]，共 256 个字符，包括英文大小写字母、数字，以及空格、回车等，如下面例子所示。

```
IN [2]:import string
       letters = string.ascii_lowercase
       s = list(map(ord, letters))
       b = list(map(bin,s))
       print(list(zip(letters, b, s)))
OUT [2]:[('a', '0b1100001', 97), ('b', '0b1100010', 98), ('c', '0b1100011', 99), ('d', '0b1100100',
100), ('e', '0b1100101', 101), ('f', '0b1100110', 102), ('g', '0b1100111', 103), ('h', '0b1101000',
104), ('i', '0b1101001', 105), ('j', '0b1101010', 106), ('k', '0b1101011', 107), ('l', '0b1101100',
```

```
108), ('m', '0b1101101', 109), ('n', '0b1101110', 110), ('o', '0b1101111', 111), ('p', '0b1110000',
112), ('q', '0b1110001', 113), ('r', '0b1110010', 114), ('s', '0b1110011', 115), ('t', '0b1110100',
116), ('u', '0b1110101', 117), ('v', '0b1110110', 118), ('w', '0b1110111', 119), ('x', '0b1111000',
120), ('y', '0b1111001', 121), ('z', '0b1111010', 122)]
```

上述例子列出了所有的小写字母，每个小写字母都对应一个数值，这个数值在计算机中是以二进制存储的，如字符'a'对应的数值是 97。有时我们会用 97 这个十进制数表示 a 在 ASCII 中的编码，使用数字表示字符，如下面例子所示。

```
IN [3]:d = range(60,70)
       s = map(chr, d)
       print(list(d))
       print(list(s))
OUT [3]:[60, 61, 62, 63, 64, 65, 66, 67, 68, 69]
        ['<', '=', '>', '?', '@', 'A', 'B', 'C', 'D', 'E']
```

因此，任何一个字符串的元素都用一个二进制表示的数据存储。如果对于非英文语言，一字节表示字符就不够用了，就需要用多字节表示。具体用几字节表示字符，国际标准组织制定了 Unicode 编码，每个字符（不管是中文、日文、韩文、俄文，还是阿拉伯文）都用至少 2 字节表示，但是这种方法对于英文字符来讲有点浪费，因此就提出了 UTF-8 编码，兼容 ASCII 编码，并得到广泛应用。我们国家也制定了相应的编码，如 GBK、GB2312、BIG5 等。这些编码只能在国内使用，国际上还是使用 UTF-8 编码。下面介绍几个常用的编码格式。

计算机字符表示的常见编码方式有以下 4 种。

- ASCII 码：ASCII（American Standard Code for Information Interchange，美国标准信息交换代码）是基于拉丁字母的编码，用 8 位（一字节）表示字符符号，即 ASCII 码最多只能表示 256 个符号。
- GBK 编码：为 GB2312 编码的扩展，它规定一个中文为两字节。
- Unicode 编码：又称作万国码，即国家之间使用计算机，无论是 GBK 编码，还是日本的 JIS 编码等，都无法同时进行编码互通，容易出现乱码等问题，因此使用 Unicode 包含了各种国家的编码方式，它规定一个字符用两个或者两个以上的字节表示。
- UTF-8 编码：它是 Unicode 编码的压缩和优化。由于 Unicode 规定一个字符用两个或者两个以上的字节表示，英文系国家使用 ASCII 码（即用一字节表示字符），因此使用 Unicode 编码会浪费许多存储空间，故制定了 UTF -8 编码，它规定一个英文字符用一字节表示，UTF-8 编码中的一个中文字符用三字节表示。

Python 语言默认使用的 Unicode 编码，在旧版本 Python 需要前导符 u 定义，Python 3 之后的版本已经不需要，写上与否没有关系。如果需要转换为其他编码，则需要进行显式的转换。我们可以通过 sys 模块的方法获取当前编码方法，如下所示。

```
IN [4]:import sys
       print(sys.stdout.encoding)
       print(sys.getdefaultencoding())
OUT [4]:UTF-8
       utf-8
```

Python 支持两种字符串类型，默认的字符串类型是 str，但是在网络数据传输、读写磁盘文件等很多场景，程序需要按照字节进行读写，这时就需要使用 Python 的 bytes 字符串类型，严格来讲不能称为字符串，它是一种字符数组。Python 严格区分这两种字符类型，它们不可以相互混用。

Python 中，bytes 数据和 str 数据之间转换就是编码/解码的过程，需要指定编码格式。

```
IN [5]:s = "东北大学"
       print(s, type(s))
       b = bytes(s, encoding = "utf-8")
       print(b, type(b))
       s2 = str(b, encoding = "utf-8")
       print(s2, type(s2))
OUT [5]:东北大学 <class 'str'>
        b'\xe4\xb8\x9c\xe5\x8c\x97\xe5\xa4\xa7\xe5\xad\xa6' <class 'bytes'>
        东北大学 <class 'str'>
```

可以看到，s 变量表示一个 UTF-8 编码的字符串，内嵌函数 str 和 bytes 可以定义对应的字符串，并可以指定编码。当然，使用前导符 b 也可以定义一个新的 bytes 字节流类型，如下所示。

```
IN [6]:sb = b'Northeastern University'
       print(sb, type(sb))
       su = str(sb)
       print(su, type(su))
OUT [6]:b'Northeastern University' <class 'bytes'>
        b'Northeastern University' <class 'str'>
```

若直接定义 bytes 字符流类型，则不可以直接用 b 前导符号定义包含**非 ASCII 编码字符**的 bytes 类型，否则就会出现错误。需要注意的是，bytes 是一种单字节的不可变数组，而 str 是一种多字节的不可变数组，其实还有一种称为 bytearray 类型的数组，它是一种单字节的可变数组，应用比较少。

如果不用上述定义方式，str 和 bytes 之间可以通过字符串对象的自带方法进行转换，如下所示。str 具有 encode 方法，bytes 具有 decode 方法，它们可以相互转换。

```
IN [7]:s = '东北大学'
       se = s.encode(encoding = "utf-8")
       print(se, type(se))
       bd = se.decode(encoding = "utf-8")
       print(bd, type(bd))
OUT [7]:b'\xe4\xb8\x9c\xe5\x8c\x97\xe5\xa4\xa7\xe5\xad\xa6' <class 'bytes'>
        东北大学 <class 'str'>
```

有时我们定义字符串数据类型的时候会包含一些引号或者不能打印出的字符，Python 里需要对这些字符进行转义处理，如以下示例所示。如果不同的引号交叉存在，如下例第一个定义❶，那么可以直接将内部引号作为字符处理，而不是作为字符串定义的标志。如果是相同的引号，如下例中第二个定义❷，那么必须将非定义标志的引号进行转义，Python 进行转义时用反斜杠表示。如果使用三引号（三单引号或三双引号）定义，如下例中的第三个定

义❸，那么三引号内的所有内容都作为注释，如果有转义应用转义，任何单引号和双引号，不管是不是成对，内部所有字符都会作为普通字符处理，因此，有时三引号会作为程序说明文档应用，后面章节会详细说明。如果在定义的前面加上前导符 r，那么和上面三引号的作用基本一样，将内部所有的内容作为字符处理，唯一不同的是，具有 r 的字符串不考虑是否转义，将转义符号也作为字符处理，如下面的第四种定义形式❹。

```
IN [8]:s = "Northeastern University's"❶
       print(s)
       s = 'Northeastern University\'s'❷
       print(s)
       s = '''Northeastern University\'s'''❸
       print(s)
       s = r'Northeastern University\'s'❹
       print(s)
OUT [8]:Northeastern University's
        Northeastern University's
        Northeastern University's
        Northeastern University\'s
```

Python 转义符列表及含义如表 5-1 所示。

表 5-1 Python 转义符列表及含义

转 义 符	含 义
\ 在行尾时	续行符号(可以直接在下一行编辑，输出时是一行)
\\	反斜杠
\'	单引号
\"	双引号
\a	响铃
\b	退格
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符(如 Excel 操作中)
\r	回车
\f	换页

5.2 掌握基本处理方法

Python 字符串提供了丰富的内嵌操作方法。首先，字符串是一种序列数据，因此索引、切片操作和序列是一样的，并且字符串也支持算术操作，如乘法、加法、in 成员判断、not 否定操作等，前面章节已经介绍，这里就不赘述。此外，字符串还提供了查找匹配、转化、内容判断和格式化四大类方法。关于字符串查找匹配，后面会讲解，本节重点介绍其他三类方法。

5.2.1 字符串处理转化

由于字符串是不可变对象，因此所有字符串的处理转化都会返回一个新的字符串对象，这一点要牢记。字符串处理转化一般是对字符串进行大小写处理、空格处理，以及子串分离、合并、替换等处理。str 对象提供的几种常用大小写转换应用方法如下面例子所示。

```
IN [9]:s = "northeastern University"
       print(s.capitalize())
       print(s.lower())
       print(s.swapcase())
OUT [9]:Northeastern university
        northeastern university
        NORTHEASTERN uNIVERSITY
```

这些方法直接调用即可，都会返回一个新的字符串，其他大小写转换方法还有 upper（转换为大写）、casefold（关闭大小写）、title（串首大写）等。字符串数据中经常会出现空格（制表符\t），有时为了处理方便，会将字符串前后的这些空格删除，比如输入用户名时不小心在行尾输入了空格，如果不处理，就会出现问题。字符串提供了：strip、lstrip 和 rstrip 三种处理方法，示例如下。

```
IN [10]:s = "\tNortheastern University\t"
        print(s.strip(),"END")
        print(s.lstrip(),"END")
        print(s.rstrip(),"END")
OUT [10]: Northeastern University END
        Northeastern University END
            Northeastern University END
```

除查找匹配，字符串最常用的就是替换和解析了。Python 提供了多种字符串替换和解析方法，下面我们看看解析方法 split、join 等。给出字符串 course，获取所有的内容缩写，取出预期结果：PA-PB-PD-PG-PI-PL-PM-PP-PS-PT-PW，将字符连接起来，并按照字母顺序排序。

```
IN [11]:course = '''
         ================ Content ===============
         + 01 --------- Python Basis language PB +
         + 02 -------- Python Advanced topics PA +
         + 03 ------- Python Standard Library PL +
         + 04 -------- Python GUI programming PG +
         + 05 -------- Python Web development PW +
         + 06 --------- Python Design pattern PD +
         + 07 ------ Python Science computing PS +
         + 08 -------- Python Text processing PT +
         + 09 ------- Python Machine learning PM +
         + 10 ------- Python Image processing PI +
         + 11 ------- Python Project practice PP +
         '''
        from functools import reduce
```

```
        text = course.strip().split('\n')
        text = [line.strip()[-4:-2] for line in text]
        text = list(filter(None, text))[1:]
        text.sort()
        print('-'.join(text))
OUT[11]:PA-PB-PD-PG-PI-PL-PM-PP-PS-PT-PW
```

上述例子使用了 split 和 join 方法，字符串还提供了其他一些方法，包括 partition、rpartition、rsplit、splitlines，当然也有一些将字符串中某部分子串进行替换的方法，包括 maketrans、replace、translate，其中 translate 方法非常适合一次替换多个字符内容的场景，而 replace 只能一次替换一个，如以下例子所示。这个例子中，原始字符串中包含了一些非字母字符，通过一个映射表自动查找并替换，简单、快速。

```
IN[12]:remap = {
            ord('\t') : " ",
            ord('\f') : " ",
            ord('\r') : None
        }
        remap2 = str.maketrans("Python","I'm007")
        s = 'Python\fis\tawesome\r\n'
        s1 = s.translate(remap)
        s2 = s.translate(remap2)
        print(s1)
        print(s2)
OUT[12]:Python is awesome

         I'm007
         is        awesOme
```

需要注意的是：这里的替换是一种字符替换，当然也可以通过 maketrans 方法制作一个映射表。如果使用 replace 方法，就需要调用很多次，效率稍微有点低，但 replace 可以对子串进行替换，这是 translate 方法无法实现的。

5.2.2 字符串格式化

字符串格式化操作在 Python 编程中非常灵活，除 Python 字符串自带的内嵌特定方法，如 ljust、zfill 等外，还提供了百分号占位符方式、format 格式，string 模块的 Template 模板也提供了类似的方法。下面对这四种方法进行介绍。

1. 内嵌格式化方式

Python 字符串 str 对象提供了以下几个固定格式化的操作方法，包括 ljust、rjust、zfill、center 和 expandtabs，下面的例子展示了这几个格式化方法的使用和效果。

```
IN[13]:s = "Northeastern University\tis beautiful"
        print("Len of s:", len(s))
        print(s.ljust(40,"-"))
        print(s.rjust(40,"-"))
        print(s.center(40,"-"))
```

```
        print(s.zfill(40))
        print(s.expandtabs())
        print(help(s.zfill))
OUT [13]:Len of s: 36
         Northeastern University is beautiful ----
         ---- Northeastern University is beautiful
         -- Northeastern University is beautiful --
         0000Northeastern University is beautiful
         Northeastern University is beautiful
```

前面三个方法可以将字符串自动增补到设定长度，增补的内容也可以设定，例子用“-”补充，如果设定长度小于字符串长度，那么不会做任何操作。方法 zifill 是在字符串左侧填充 0，expandtabs 是将字符串中所有的制表符统一用空格替代。

2. 百分号占位符方法

该方法在很多其他语言中也比较常见，一般表示形式如图 5-1 所示，通过百分号对要填充的内容进行格式化占位，然后用实际的数据替换。占位符方式在早期 Python 中应用比较广泛，但是在 Python 3 版本后，format 方式应用得更多一些。

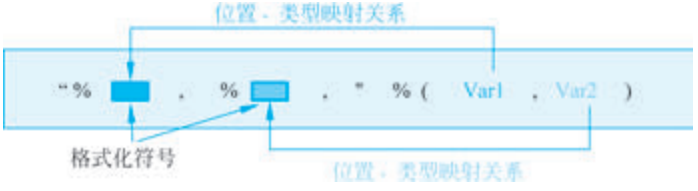

图 5-1 百分号占位符格式化形式

字符串格式化输出有很多用途，如格式化输出数据到终端时更容易阅读，网页模板进行动态内容替换等。Python 提供的一些格式化类型见表 5-2。

表 5-2 格式化类型

符　号	说　明
%c	格式化字符及其 ASCII 码
%s	格式化字符串，获取传入对象的__str__方法的返回值
%r	格式化字符串，获取传入对象的__repr__方法的返回值
%d	格式化整数，将整数、浮点数转换成十进制数表示
%u	格式化无符号整型，将整数、浮点数转换成无符号十进制数表示
%o	格式化无符号八进制数，将整数转换成八进制数表示
%x	格式化无符号十六进制数，将整数转换成十六进制数表示
%X	格式化无符号十六进制数(大写)
%f	格式化浮点数字，可指定小数点后的精度，将整数、浮点数转换成浮点数表示
%e	用科学记数法格式化浮点数，将整数、浮点数转换成科学记数法
%E	作用同%e，用科学记数法格式化浮点数，将整数、浮点数转换成科学记数法
%g	%f 和%e 的简写，自动转换，将整数、浮点数转换成浮点型或科学记数法表示
%G	%f 和%E 的简写，自动转换，将整数、浮点数转换成浮点型或科学记数法表示

需要注意的是：如果想在字符串中保留格式化标志，需要用 %%表示一个百分号。此外，Python 中百分号格式化是不存在自动将整数转换成二进制表示方式的。下面举几个例子，以便于理解。

```
IN [14]:name = "Northeastern University"
        num = 3200
        print("Hi, %40s is beautiful!" % (name))
        print("There are %d staffs." % (num))
OUT [14]:Hi, Northeastern University is beautiful!
         There are 3200 staffs.
```

通过上面例子可以看出，字符串的格式化输出使得字符串的使用更加灵活且格式输出一致。占位符的接收方式有两种，其格式不一样，分别是：元组、字典。上面例子是使用元组的情况，下面的字典例子是上面例子的改写。

```
IN [15]:data = {"uni" : "Northeastern University",
                "num" : 3200}
        print("Hi, %(uni)30s is beautiful!" % data)
        print("There are %(num)d staffs." % data)
OUT [15]:Hi, Northeastern University is beautiful!
         There are 3200 staffs.
```

3. format 格式化方法

这种方法在 Python 3 版本以后应用最为广泛，也是最灵活的一种方式。该方式是字符串的内嵌方法，与它类似的还有一个方法 format_map。格式化方法 format 也支持位置映射、关键字映射，如图 5-2 所示的位置映射和关键词映射方式。

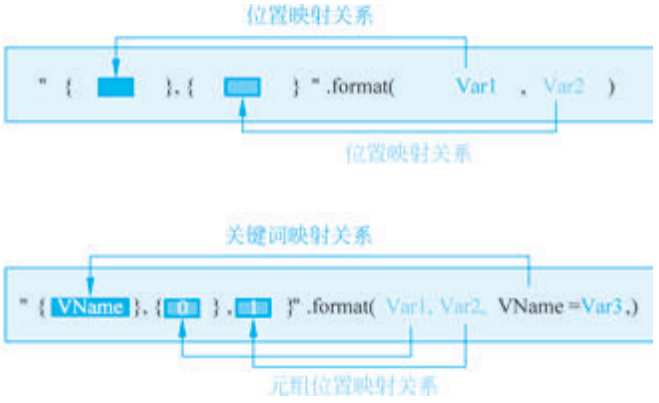

图 5-2 格式化 format 方法形式

同样，上面的例子可以用 format 方法改写，如下面例子。

```
IN [16]:name = "Northeastern University"
        num = 3200
        print("Hi, {0} is beautiful!".format(name))
        print("There are {0} staffs.".format(num))
OUT [16]:Hi, Northeastern University is beautiful!
         There are 3200 staffs.
```

如果按照默认数据次序，上面{}里面的元素索引序号也可以省略。format 格式化方法使用关键词映射，参见下面例子，将关键词名称写到{}里面即可。

```
IN [17]:name = "Northeastern University"
        num = 3200
        print("Hi, {uni} is beautiful!".format(uni = name))
        print("There are {num} staffs.".format(num = num))
OUT [17]:Hi, Northeastern University is beautiful!
         There are 3200 staffs.
```

format 格式化方法也支持传入列表中元素的访问，以及规定传入参数的格式，如下面例子所示。

```
IN [18]:name = dict(zip(("Fst","Sec"),"Northeastern\
                    University".split()))
        num = [3200,3400]
        print("Hi, {name[Fst]} is beautiful!"\
                          .format(name = name))
        print("There are {0[1]} staffs.".format(num))
        print("{} - {} - {}".format("NEU","CSE","PYTHON"))
        print("{2} - {1} - {0}".format("NEU","CSE","PYTHON"))
        print("{cname} - {course} - {uname}"\
            .format(course = "PYTHON",\
                    uname = "NEU",cname = "CSE"))
        print("{0[1]} - {0[0]} - {0[2]} - {1}".format(("NEU","CSE","PYTHON"),'Liwei'))
OUT [18]:Hi, Northeastern is beautiful!
         There are 3400 staffs.
         NEU - CSE - PYTHON
         PYTHON - CSE - NEU
         CSE - PYTHON - NEU
         CSE - NEU - PYTHON - Liwei
```

对 format 结果进行填充对齐和数值精度设置，如下例所示。

```
IN [19]:print("{0} * {1} = {2:0 > 2}".format(3,2,2 * 3))
        print("{: * ^30}".format('centered'))
        print("{:.3f}".format(2.1415))
        print("{:.10f}".format(3.1415))
        data = "{:s},{:d},{:b},{:o},{:x},{:X},{:e},{:.2f}"
        print(data.format("hello",15,15,15,15,15,16.6666,16.3333))
OUT [19]:3 * 2 = 06
         ********** centered **********
         2.142
         3.1415000000
         hello,15,1111,17,f,F,1.666660e + 01,16.33
```

上面最后一个格式化语句中列出了多种数据类型，s 表示占位符字符串，d 表示十进制，b 表示二进制，o 表示八进制，x 表示十六进制，X 表示大写十六进制，e 表示科学记数法，.2f 表示浮点数，并保留两位小数。填充对齐符号包括^、<、>，分别表示居中、左对齐、右对齐。

当然，也可以使用字符串方法 format_map，和 format（** mapping）用法类似，它们的不同之处在于 mapping 会被直接使用，而不是复制到一个 dict。使用此方法的一个例子是当 mapping 为 dict 的子类时，如下例所示。

```
IN [20]:info = "my name is {name},I'am {age} years old."
        res = info.format_map({"name":"John","age":22})
        print(res)

        class Default(dict):
            def __missing__(self, key):
                return key
        res = '{name} was born in {country}'.format_map\
                            (Default(name = 'John'))
        print(res)
OUT [20]:my name is John,I'am 22 years old.
         John was born in country
```

传入的数据可以直接用一个字典，也可以是一个派生的子类对象。下例根据上面的格式化方法，演示一个打印的加法表。

```
IN [21]:for i in range(1,10):
            start = 1
            while start <= i:
                print("{0} + {1} = {2:0 > 2}".format(\
                        start, i, start + i), end = "\t")
                start += 1
            print()
OUT [21]:1 + 1 = 02
         1 + 2 = 03   2 + 2 = 04
         1 + 3 = 04   2 + 3 = 05   3 + 3 = 06
         1 + 4 = 05   2 + 4 = 06   3 + 4 = 07   4 + 4 = 08
         1 + 5 = 06   2 + 5 = 07   3 + 5 = 08   4 + 5 = 09   5 + 5 = 10
         ……
```

4. string 对象格式化

模块 string 中的 Template 类将一个字符串数据设置为模板，通过替换变量的方法，最终得到想要的格式文本数据。和前面的百分号占用和 format 方式相比，它简化了特定的字符串置换操作。

模块 string. Template 是利用 $ 符号进行“关联”，用 substitute 方法取值的。利用 string. Template 方法不需要考虑参数的数据类型。这是 string. Template 方法与前面两种方法最重要的不同之处。Template 方法直接将参数转换为字符串格式，然后将转换后的字符串直接插入结果中。但并没有可用的格式化选项供我们选择，如有数值数据，对于一个浮点数（如上述例子所示）来讲，我们没办法控制代表这个浮点数数值的位数，如下面例子所示。

```
IN [22]:from string import Template

        temp = Template("""
        <html>
            <div>Hi, $name</div>
            Please say hello word,
            <div>$where</div>
        </html>""")

        data = {'name':"Liwei", "where":"NEU"}

        print("TEMPLATE:", temp.substitute(data))
OUT [22]:TEMPLATE:
         <html>
             <div>Hi, Liwei</div>
             Please say hello word,
             <div>NEU</div>
         </html>
```

上例中我们利用 substitute 取值，如果 $ 关联的字符串在前面定义的 where 中不存在怎么办？为了避免上述问题的产生，我们利用 safe_substitute 方法取值，读者可以跟原生的 substitute 方法对比一下。safe_substitute 方法可以避免这个错误，保证程序的流畅性。

实际中，大家可能习惯利用%操作符进行字符串的置换，那么，如果既想利用 string. Template 方法的便捷性，又想按照自己的意愿与需求定义额外的功能，这就需要我们新定义一个继承自 string. Template 的类，在这里修改其中的某些属性以满足我们的需求。下面例子中，MyTemplate 类继承自 string. Template，修改了操作符 delimiter 与 id 模式 idpattern，实现了利用%关联代表 key 的字符串，然后利用正则表达式使 safe_substitute() 只能匹配出带下画线的且由 a～z 组成的字符串。

```
IN [23]:import string

        class MyTemplate(string.Template):
            delimiter = '%'
            idpattern = '[a-z]+_[a-z]+'

        template_text = '''
        Delimiter : %%
        Replaced : %with_underscore
        Ignored : %notunderscored
        '''
        d = {
            'with_underscore':'replaced',
            'notunderscored':'not replaced'
        }

        t = MyTemplate(template_text)
        print('Modified ID pattern:')
        print(t.safe_substitute(d))
```

```
OUT [23]:Modified ID pattern:

         Delimiter : %
         Replaced : replaced
         Ignored : %notunderscored
```

上例中，由于代表 key 的字符串‘notunderscored’没有下画线，没有匹配到，所以结果中只能得出%notunderscored，不能取到具体的值。需要注意的是：这种方法在实际中经常使用。实际中，我们需要根据具体的需求灵活更改模块中某个对象的某个属性以满足具体的需求。

上面讲述了各种字符串格式化的方法，当然还有一些特殊的用法，如使用前缀 f，需要读者参考官方标准的文档，如下这些用法也是经常遇到的。

```
IN [24]:name = {'Fst': 'Northeastern', 'Sec': 'University'}
        print("{{0}}".format("can't see me"))
        print(format(10.0, "8.2g"))
        print(f"hello, {name}")
OUT [24]:{0}
              10
         hello, {'Fst': 'Northeastern', 'Sec': 'University'}
```

5.2.3 字符串内容判断

字符串包含的内容判断在很多程序设计中经常用到。Python 内嵌的字符串内容判断方法有很多，如表 5-3 所示。

表 5-3 Python 内嵌的字符串内容判断方法

方 法	说 明	方 法	说 明
startswith	起始判定	endswith	终止判定
isalnum	字母数字	isalpha	字母
isascii	ASCII 字符	isdecimal	数据
isdigit	数字	isidentifier	标识符
islower	小写	isnumeric	数值
isprintable	可打印	isspace	空格
istitle	首字母大写	isupper	大写

这些方法调用非常简单，返回逻辑值，如下面例子所示，就不一一举例了。

```
IN [25]:fnames = ['Makefile', 'foo.c', 'bar.py', 'spam.c', 'spam.h']
[name for name in fnames if name.endswith(('.c', '.h'))]
OUT [25]:['foo.c', 'spam.c', 'spam.h']
```

5.3 字符串匹配和查找

上面讲述了字符串的基本处理方法，字符子串查找/匹配在程序应用中占据重要地位，特别是文本数据分析应用中的预处理过程，存在大量的数据清洗工作。Python 除提供内嵌

的简单查找方法外，也提供了正则表达式功能模块 re。

Python 提供的内嵌查找方法有 index、find 等，对于不太复杂的数据查找需求是比较快速的，如下例所示。

```
IN [26]:s = "东北大学位于东北地区,沈阳东北大马路"
        print("北大" in s)
        print(s.find("北大"),s.rfind("北大"))
        print(s.index("东北"),s.rindex("东北"))
        print(s.index("东北",5),s.rindex("东北",6, 10))
        print("size:", s.count("东北"))
OUT [26]:True
         1 14
         0 13
         6 6
         size: 3
```

但是，对于复杂一点的查找需求，上述这些方法可能就满足不了，并且效率不够高，特别是数据规模比较大的情况下。这时就要思考使用 re 模块处理文本。下面首先介绍正则表达式的概念，然后介绍 Python 中支持的正则表达式功能，最后列举几个应用示例。

5.3.1 正则表达式基础

正则表达式(regular expression)精确定义了字符串匹配的模式(pattern)，用于子串查找、子串替换等任务场景。例如：

- neue+r 模式可以匹配 neueer、neueeer、neueeeer 等，+ 号代表前面的字符必须至少出现 1 次(1 次或多次)。
- neue *r 模式可以匹配 neur、neuer、neueeeeer 等，* 号代表字符可以不出现，也可以出现 1 次或者多次(0 次或 1 次或多次)。
- neue?r 模式可以匹配 neur 或者 neuer，? 问号代表前面的字符最多只可以出现 1 次(0 次或 1 次)。

构成正则表达式的元素可以是单个字符、字符集合、字符范围、字符间的选择或任意元素组合。Python 的正则表达式也是由普通字符(如字符 a～z)以及特殊字符(称为“元字符”)组成。

(1) 普通字符

普通字符包括所有大写和小写字母、所有数字、所有标点符号和一些其他符号。

(2) 非打印字符

非打印字符也可以是正则表达式的组成部分，如表 5-4 列出了转义字符及其描述。

表 5-4 转义字符及其描述

字　符	描　述
\cx	由 x 指明的控制字符
\f	换页符，等于 \x0c 和 \cL
\n	换行符，等于 \x0a 和 \cJ
\r	回车符，等于 \x0d 和 \cM

续表

字　符	描　述
\s	任何空白字符，包括空格、制表符、换页符等
\S	任何非空白字符，等于[^\f\n\r\t\v]
\t	一个制表符，等于 \x09 和 \cI
\v	一个垂直制表符，等于 \x0b 和 \cK

(3) 特殊字符

特殊字符是一些有特殊含义的字符，如上面说的 neue * r 中的 * 。若正好需要匹配特殊字符，则使用反斜杠进行转义。表 5-5 列出了特殊字符及其描述。

表 5-5　特殊字符及其描述

特 殊 字 符	描　述
$	输入字符串的结尾位置
( )	标记一个子表达式的开始位置和结束位置
*	前面的子表达式出现零次或多次
+	前面的子表达式出现一次或多次
.	除换行符 \n 外的任何单字符
[	标记一个中括号表达式的开始
?	前面的子表达式出现零次或一次，或指明一个非贪婪限定符
\	转义符，'\\' 匹配 "\"
^	输入字符串的开始位置
{	标记限定符表达式的开始
|	两项之间的一个选择

(4) 限定符

限定符确定正则表达式给定的一个元素出现多少次。表 5-6 列出了正则表达式的限定符。

表 5-6　正则表达式的限定符

限　定　符	描　述
*	前面的子表达式出现零次或多次
+	前面的子表达式出现一次或多次
?	前面的子表达式出现零次或一次
{n}	n 是一个非负整数，确定的 n 次
{n,}	n 是一个非负整数，至少 n 次
{n,m}	m 和 n 均为非负整数，其中 n≤m，最少匹配 n 次且最多匹配 m 次

限定符 * 和 + 默认都是贪婪匹配的，会尽可能多地匹配文字，只有在它们的后面加上一个 ? 才能实现非贪婪或最小匹配。如下实例代码匹配 h1 标签内容，数据如下。

<h1>NEU-东北大学</h1>

- 贪婪写法：表达式<.*>匹配从开始小于符号(<) 到关闭 h1 标记的大于符号(>) 的所有内容。
- 非贪婪写法：表达式<.*?>只匹配 h1 标签 <h1>。也可以使用正则表达式<\w+?>匹配 h1 标签。

可见，通过在 *、+ 或 ? 限定符后放置 ?，该表达式从“贪婪”表达式转换为“非贪婪”表达式或者最小匹配。还有一些正则表达式符号，如定位符、选择、反向引用等，这里不赘述，只是简单介绍背景知识，详细内容可以参考官方文档。

5.3.2 re 模块

Python 的正则表达处理模块 re，借鉴了 Perl 语言中的正则表达式模式。re 模块可以实现字符串匹配、替换、字串截取等功能，如 match、search、find、findall、sub 等。下面介绍这些函数的功能及基本使用方法。

(1) re.match 方法

函数原型 match(pattern,string,flags=0)，如果字符串中的 0 或者多个字符匹配到了正则表达式样式，就返回一个相应的匹配对象。如果没有匹配成功，就返回 None，如下面例子所示。

```
IN [27]:import re
        print(re.match('www','www.neu.edu.cn').span())
        print(re.match('cn', 'www.neu.edu.cn'))
OUT [27]:(0, 3)
         None
```

匹配结果对象总有一个布尔值，使用该布尔值判断是否获取成功，调用匹配对象的 group 方法，获取一个或者多个匹配的子组。

```
IN [28]:import re
        line = "Cats are smarter than dogs"
        matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)
        if matchObj:
            print("group() : ", matchObj.group())
            print("group(1) : ", matchObj.group(1))
            print("group(2) : ", matchObj.group(2))
        else:
            print ("No match!!")
OUT [28]:group() : Cats are smarter than dogs
         group(1) : Cats
         group(2) : smarter
```

(2) re.search 方法

和 match 检查字符串开头不同，search 检查字符串的任意位置(默认 Perl 中的行为)，函数原型是 search(pattern,string,flags=0)。扫描整个字符串，找到匹配样式的第一个位置，并返回一个相应的匹配对象。如果没有匹配，就返回一个 None，如下面示例所示。

```
IN [29]:import re
        print(re.search('www','www.neu.edu.cn').span())
        print(re.search('cn','www.neu.edu.cn').span())
OUT [29]:(0, 3)
         (13, 15)
```

同样，也可以使用 group 进行分组处理，使用 search 返回对象，如下所示。

```
IN [30]:import re
        line = "Cats are smarter than dogs"
        matchObj = re.search( r'(.*) are (.*?) .*', line, re.M|re.I)
        if matchObj:
            print("group() : ", matchObj.group())
            print("group(1) : ", matchObj.group(1))
            print("group(2) : ", matchObj.group(2))
        else:
            print ("No match!!")
OUT [30]:group() : Cats are smarter than dogs
         group(1) : Cats
         group(2) : smarter
```

(3) 检索和替换

re 模块使用 sub 函数实现字符串替换功能，函数原型为 sub(pattern,repl,string,count=0,flags=0)，返回通过使用 repl 替换在 string 最左边非重叠出现的 pattern 而获得的字符串。如果样式没有找到，则不加改变地返回 string。repl 可以是字符串或函数；如 repl 为字符串，则其中的任何反斜杠转义序列都会被处理，如下面实例所示。

```
IN [31]:import re
        phone = "2004-959-559 # 这是一个电话号码"
        num = re.sub(r'#.*$', "", phone)
        print("电话号码是:", num)
        num = re.sub(r'\D', "", phone)
        print("电话号码是:", num)
OUT [31]:电话号码是: 2004-959-559
         电话号码是: 2004959559
```

(4) findall 方法

re 模块的 findall 函数实现根据正则表达式查找所有匹配的子串功能。函数原型为 findall(pattern,string,flags=0)，以字符串列表或字符串元组列表的形式返回 pattern 在 string 中的所有非重叠匹配。对 string 的扫描从左至右，匹配结果按照找到的顺序返回。空匹配也包括在结果中，如下面实例所示。

```
IN [32]:import re
        p = re.compile(r'\d+')
        result1 = p.findall('baidu 123 google 456')
        result2 = p.findall('neu123mic888Wei456', 0, 10)
        print(result1)
        print(result2)
OUT [32]:['123', '456']
         ['123', '8']
```

(5) re.split 方法

re 模块的 split 函数实现按照多个分隔符进行字符串切分，比 str 对象的 split 功能强大一些。str 对象的 split 方法只能接受一个分隔符，re 的 split 可以按照正则表达式规则书写多个分隔符。split 函数的原型为 split(pattern,string,maxsplit=0,flags=0)，用 pattern

分开 string。如果在 pattern 中捕获到括号，那么所有组里的文字也会包含在列表里。如果 maxsplit 非零，最多进行 maxsplit 次分隔，剩下的字符全部返回到列表的最后一个元素，如以下示例所示。

```
IN [33]:import re
        line = 'asdf fjdk; afed, fjek,asdf, foo'
        re.split(r'[;,\s]\s*', line)
OUT [33]:['asdf', 'fjdk', 'afed', 'fjek', 'asdf',
         'foo']
```

回到 5.2.1 节中的例子，提取 course 字符串中所有的缩写，如果使用 re 正则表达式，将会容易得多，同时我们还可以快速提取出所有的数字，如以下示例所示。

```
IN [34]:import re
        text = re.findall(r"(P\w)\s\+",course)
        text.sort()
        print("-".join(text))
        res = re.findall(r"(\d{2})", course)
        print(res)
OUT [34]:PA-PB-PD-PG-PI-PL-PM-PP-PS-PT-PW
         ['01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11']
```

正则表达式在网页解析中的应用比较多，虽然有很多好用的第三方库，如 beautifulsoup 等。这里演示一个简单的处理例子。从 http://www.neu.edu.cn 获取网页数据，并保存为 neu.html 文件。如果想获取该网页中所有的图片链接，可以通过字符串查找方法 find，这种方法非常烦琐，首先需要查找图片标签的开始位置和结束位置，然后使用字符串切片操作获取子串。当然，也可以使用简便的 re 模块，如以下示例所示。

```
IN [35]:import re
        neu = open("neu.html",encoding="utf-8").read()
        res =re.findall(r"<img\s(src=\'.*\.\w{3}\')\s.*/>", neu)
        print(res)
OUT [35]:
["src='/_upload/article/images/a9/57/6449931f44a2962388119aab7376/8833fd51-ca8a-4e5f-
aa09-04759a07c5eb_s.png'",
"src='/_upload/article/images/41/7a/9d9f35944b219be0b742e55577aa/e264f44f-cc7e-4554-
b7f4-dae1ddaf0c8e_s.jpg'",
"src='/_upload/article/images/39/4d/dad2578c4c408c3477705188c161/5dafb677-f355-43e6-
a92e-866414543980_s.jpg'",
"src='/_upload/article/images/fb/01/36fad04e4e1bb0b6d52baa5630dc/dc70c9e0-e679-43e2-
a3a9-546a6ceedc22_s.jpg'"]
```

正则表达式应用比较灵活，还需要读者在实践中不断练习，才能更好地掌握。

5.4 中文文本处理

对于中文和日文这样的特殊亚洲语系文本而言，字和字之间是紧密相连的，单纯从文本形态上无法区分具备独立含义的词(英语系纯天然由空格分隔不同的 word)，而不同的词以

不同的方式排布，可以表达不同的内容和情感，因此在很多中文任务中，我们需要做的第一个处理叫作分词。

这是一个非常基础的功能，但是会较大程度地影响下游任务（机器翻译、情感分析、文本理解）的效果。目前常见的分词方法主要有以下几种。

- 基于词典匹配的分词方法（正向最大匹配法、逆向最大匹配法和双向匹配分词法等）；
- 基于统计的分词方法（如隐马模型（HMM）、条件随机场（CRF））；
- 基于深度神经网络的 RNN、LSTM 等方法。

本节主要介绍 Python 第三方模块 jieba 分词库。项目地址为 https://pypi.org/project/jieba。jieba 中文分词支持的三种分词模式如下。

- 精确模式：按照语义将句子中的词准确地切分，适合文本分类和语义分析场景。
- 全模式：将句子中所有的可能成为词的词语尽可能都提取出来，不能解决歧义问题，但把所有可能的语义词都提取出来了。
- 搜索引擎模式：在精确模式分词结果基础上，进一步对长词再次进行切分，取得更多的词汇，提高搜索引擎的命中率和召回率。

（1）分词

方法 cut 和 cut_for_search 都返回一个可迭代的生成器对象。用户使用 for 循环即可获取所有分词词语。其中据官方文档介绍，两个函数的原型如下所示。

方法 cut(self,sentence,cut_all=False,HMM=True,use_paddle=False)，4 个参数：sentence 是待分词的字符串；cut_all 用来控制是否采用全模式；HMM 用来控制是否使用 HMM 模型；use_paddle 用来控制是否使用 paddle 模式下的分词模式，paddle 模式采用延迟加载方式，如以下示例所示。

```
IN [36]:import jieba
        r = jieba.cut("中华人民共和国东北大学栗伟计算机学院")
        for w in r:
            print(w)
OUT [36]:中华人民共和国
        东北大学
        栗伟
        计算机
        学院
```

方法 cut_for_search 的原型为 cut_for_search(self,sentence,HMM=True)，它接收两个参数：需要分词的字符串 sentence；是否使用 HMM 模型。该方法适用于搜索引擎构建倒排索引的分词，粒度比较细，如以下示例所示。

```
IN [37]:import jieba
        d = "中华人民共和国东北大学栗伟计算机学院"
        r = jieba.cut_for_search(d)
        print("/".join(r))
OUT [37]:中华/华人/人民/共和/共和国/中华人民共和国/东北/北大/大学/东北大学/栗伟/计算/
算机/计算机/学院
```

（2）载入词典

用户可以指定自己自定义的词典，扩充 jieba 词库里没有的词。该库提供载入用户词典的 load_userdict(file_name)，其中 file_name 为文件类对象或自定义词典的路径。方法要求输入文件一个词占一行，包括词语、词频（可省略）、词性（可省略），中间用空格隔开。

（3）词性标注

该库提供 jieba.posseg.POSTokenizer(tokenizer=None)方法自定义分词器，默认使用 jieba.Tokenizer 分词器。方法 jieba.posseg.dt 是词性标注分词器。词性标注采用和 ictclas 兼容的标记法。使用方法如以下示例所示。

```
IN [38]:import jieba.posseg as pseg
        d = "中华人民共和国东北大学栗伟计算机学院"
        for w,f in pseg.cut(d):
            print(w,f)
OUT [38]:中华人民共和国 ns
        东北大学 nt
        栗伟 nr
        计算机 n
        学院 n
```

词性标签列表如表 5-7 所示。

表 5-7　词性标签列表

标　签	含　义	标　签	含　义	标　签	含　义	标　签	含　义
n	普通名词	f	方位名词	s	处所名词	t	时间
nr	人名	ns	地名	nt	机构名	nw	作品名
nz	其他专名	v	普通动词	vd	动副词	vn	名动词
a	形容词	ad	副形词	an	名形词	d	副词
m	数量词	q	量词	r	代词	p	介词
c	连词	u	助词	xc	其他虚词	w	标点符号
PER	人名	LOC	地名	ORG	机构名	TIME	时间

当然，jieba 模块还有其他的功能，如关键词抽取，根据需要调用即可，这里就不一一举例说明。

5.5　字符串的相似性比较

字符串的相似性比较应用场合很多，像拼写纠错、文本去重、上下文相似性等。评价字符串相似度最常见的办法是：把一个字符串通过插入、删除或替换这样的编辑操作变成另一个字符串所需要的最少编辑次数，这种计算方法就是编辑距离（edit distance）度量方法，也称为 Levenshtein 距离。其他常用的度量方法还有 Jaccard distance、J-W 距离（Jaro-Winkler distance）、余弦相似性（cosine similarity）、欧几里得距离（Euclidean distance）等。

Python 提供了标准模块 difflib 提供的类和方法，用来进行序列的差异化比较，它能比对文件并生成差异结果文本或者 html 格式的差异化比较页面，如果需要比较目录，可以使用 filecmp 模块。从数据中查找相似的目标字符串，如下面实例所示。

```
IN [39]:import difflib
        data = ['ape', 'apple', 'peach', 'puppy']
        res = difflib.get_close_matches('appel', )
        print(res)
OUT [39]:['apple', 'ape']
```

根据相似程度排序后输出结果，这在单词纠错方面比较有用。该模块方法 SequenceMatcher 也可以计算两个字符串之间的相似度，如下面代码：

```
>>> difflib.SequenceMatcher(None, 'tide', 'diet').ratio()
```

或者封装到一个函数，如以下示例所示。

```
IN [40]:import difflib
        def str_sim(s1, s2):
            return difflib.SequenceMatcher(None, s1,\
                                           s2).quick_ratio()

        print(str_sim('爱尔眼科沈阳医院', '沈阳爱尔眼科医院'))
        print(str_sim('和平区妇幼保健站', \
                      '沈阳市和平区妇幼保健站'))
        print(str_sim('广州市医院', '广东省中医院'))
OUT [40]:1.0
        0.8421052631578947
        0.5454545454545454
```

也可以简化为匿名函数去调用，如以下示例所示。

```
OIN [41]:import difflib

         cstr = lambda s1, s2: difflib.SequenceMatcher(None, s1, s2).quick_ratio()
         print(cstr("大东北的大学", "东北大学"))
         print(cstr("东北的大学", "东北大学"))
         print(cstr('和平区妇幼保健站', '沈阳市和平区妇幼保健站'))
OUT [41]:0.8
         0.8888888888888888
         0.8
```

关于 difflib 模块的功能还有很多，这里就不一一描述了，本书只需要我们了解这些基本方法。

小结

本章首先介绍了关于字符串相关的处理操作，包括字符串的 str 和 bytes 类型，以及它们之间的编码转换；然后介绍了对字符串的基本处理，特别是对字符串的查找匹配，这在很多应用中用途广泛；最后还介绍了第三方的中文分词库 jieba，以及 Python 标准库 difflib 的基本用法。

习题 5

5.1 定义变量 name=“Tom”,请用两种输出字符串为"Hello,Tom"的代码实现。

5.2 字符串 s=“a\nb\x1f\000d”,请列出 s 中包含的字符。

5.3 语句:'{:7.2f}{:2d}'.format(101/7,101%8),给出该语句的执行结果。

5.4 字符串 s = 'What is your name',给出子串 s[11:2:-2]的内容。

5.5 字符串 s="To Be or Not to Be",给出 s.strip("To Be")语句的执行结果。

5.6 关于字符串提供的查询方法,判断以下说法正确与否。

A. count 方法用于统计字符串里某个字符出现的次数。( )

B. find 方法检测字符串中是否包含子字符串 str,如果包含子字符串,则返回开始的索引值,否则报一个异常。( )

C. index 方法检测字符串中是否包含子字符串 str,如果包含 str,则不再返回-1。( )

5.7 字符串 s="abcebd",运行 s.count("a")与 s.count("a",1)的结果有什么不同?

5.8 请描述下列程序片段的运行结果。

```
import string
name, n = 'Guido', 37
s = string.Template('$name has $n messages')
s.substitute(vars())
```

5.9 字符串 s1="Northeastern",s2="University",请至少写出 4 种可以把 s1 和 s2 拼接在一起形成 s="Northeastern University"表达式的实现方法。

5.10 编写代码,实现:判断输入的字符串是否为回文字符串,例如"level""noon"均为回文字符串。

5.11 编写代码,实现:对输入字符串中大写英文字母、小写英文字母、数字以及其他字符定位并计数的功能。

5.12 请使用正则表达式实现从字符串中提取首字母缩写词,例如从"Federal Emergency Management Agency"中提取出每个单词的首字符,即"FEMA"。

5.13 请利用 5.4 节和 5.5 节的方法对“辽宁省是中国东北地区的一个省份”和“东北大学是中国的一所大学”两个字符串进行相似度比较。