第 5 章 字符串和正则表达式

CHAPTER 5

字符串是计算机语言中的一种数据类型，用于表示和操作文本数据。字符串在信息传播、数据分析、信息处理、信息安全等方面承担着重要的作用。例如，在大数据爆发时代，学生可以通过掌握字符串的相关知识，了解如何提取、清洗和分析字符串数据，从而掌握信息的真实性，并利用可靠信息进行社会问题研究和决策。字符串知识点的学习，可以提升学生的网络道德认识，加强学生的信息安全意识。

正则表达式有着强大的文本处理能力，在数据处理、信息提取、数据分析、文本处理、数据匹配和验证等方面扮演着重要的角色。例如，学生可以通过正则表达式的模式匹配功能，查找特定的文本内容，识别和抵制网络上的虚假信息，提高学生的网络信息判断技能。正则表达式知识点的学习，可以提高学生的信息素养。

学习目标

（1）理解字符串的编码，掌握字符串的创建、转义字符的使用、字符串的截取和字符串的常用方法。

（2）了解正则表达式的概述，理解正则表达式元字符的语法规则，并掌握正则表达式模块的常用方法和匹配选项编译标志。培养学生在信息时代的思维能力、社会责任感，提高学生的信息素养。

（3）通过本章课程的学习，加深学生对传统文化的认识，加强学生的信息安全意识。

学习重点

（1）掌握字符串的截取和字符串的常用方法。

（2）掌握正则表达式元字符的语法规则，并掌握正则表达式模块的常用方法。

学习难点

正确理解正则表达式元字符的语法规则和相关技术的应用。

视频讲解

5.1 字符串

字符串是 Python 程序中常用的数据类型之一，大多数 Python 程序涉及字符串处理的内容。本节将对字符串的编码方式及常用的处理方法给予详细的介绍。

5.1.1 字符串编码

字符编码的目的在于存储和传输，不同语言的字符编码，采取的编码方式不同。最

早的字符编码方式是 ASCII 码，ASCII 码采用 1 字节编码，而且只能对 127 个字符编码，即 10 个数字、26 个大小写英文字母和一些其他字符。但是，处理中文的汉字时，1 字节明显是不够的。1980 年，中国国家标准总局发布了《信息交换用汉字编码字符集》（GB/T 2312—1980），采用 2 字节表示常用的 6763 个汉字和 682 个非汉字字符。

在多语言的混合文本中，不同国家的编码方式可能会产生冲突，导致文本中出现乱码。而 Unicode 编码字符集把不同的编码方式统一到同一套编码方案里，为每种语言中的每个字符设定了统一且唯一的二进制编码，以满足跨语言、跨平台进行文本转换和处理的需求，解决了不同编码方式的冲突问题。在 Unicode 编码方案中，常用的字符用 2 字节表示，不常用的字符用 3～4 字节表示。这样的表示方式又产生了新的问题，一篇全英文字母的文章，用 Unicode 编码比 ASCII 码需要多一倍的存储空间，导致存储空间浪费、传输效率低等问题。

于是，UTF（Unicode Transformation Format，Unicode 转换格式）被提出，UTF 规定了 Unicode 字符集中各类字符的存储和传输格式，常用的格式有 UTF-8、UTF-16 和 UTF-32 等。Linux、Python 和浏览器等采用 UTF-8 编码，Windows、Java 等采用 UTF-16 编码。UTF-8 编码根据 Unicode 字符集中字符的分类不同，使其编码长度产生相应的变化，如，英文字母被编码为 1 字节，汉字被编码为 3 字节，生僻字符被编码为 4～6 字节。因此，UTF-8 编码又被称为“可变长编码”。

针对单个字符的编码，Python 提供了两个互相转换的内置函数，即 ord() 函数获取给定字符的 Unicode 编码值，chr() 函数将给定的 Unicode 编码值转换为对应的字符。示例代码如下所示。

```
>>> ord("中"), ord("1"), ord("a")    # 获取单个字符的 Unicode 编码值
(20013, 49, 97)
>>> ord("中国")                      # 不是单个字符，会引发 TypeError 异常
TypeError: ord() expected a character, but string of length 2 found.
>>> chr(20013), chr(49), chr(97)     # 返回 Unicode 编码值所对应的单个字符
('中', '1', 'a')
```

Python 语言中，字符串有 2 种不同的形式。str 类型通常用于表示 Unicode 字符序列，能够直观地阅读和理解。而 bytes 类型则将字符序列编码，并以字节序列的形式表示，主要用于字符序列的存储和传输。将 str 类型转换为 bytes 类型，可以使用 str 类型的 .encode() 方法进行转换。该方法会按照指定的字符编码将字符串转换为对应的字节序列，并返回一个新的 bytes 对象。相反地，将 bytes 类型转换为 str 类型，可以使用 bytes 类型的 .decode() 方法进行转换。该方法会按照指定的字符编码将字节序列解码为对应的字符串，并返回一个新的 str 对象。示例代码如下所示。

```
>>> "CHINA".encode("UTF-8")                          # str 类型转换为 bytes 类型
b'CHINA'
>>> "中国".encode("UTF-8")                           # str 类型转换为 bytes 类型
b'\xe4\xb8\xad\xe5\x9b\xbd'
>>> b'CHINA'.decode("UTF-8")                         # bytes 类型转换为 str 类型
'CHINA'
>>> b'\xe4\xb8\xad\xe5\x9b\xbd'.decode("UTF-8")      # bytes 类型转换为 str 类型
'中国'
```

5.1.2 字符串的创建

Python 程序中，字符串的创建方法可以分为两种。

1. 赋值法

赋值法创建字符串，使用英文单引号、英文双引号或英文三引号其中之一作为定界符进行创建，基本语法格式如下所示。

```
字符串名 =' 字符序列 '
或：字符串名 =" 字符序列 "
或：字符串名 ='''字符序列 '''
```

需要说明的是，英文单引号为定界符创建字符串时，如果字符序列中又出现单引号，则该单引号需要使用转义字符进行转义；英文双引号为定界符创建字符串时，如果字符序列中又出现双引号，则该双引号需要使用转义字符进行转义；英文三引号为定界符创建字符串时，如果字符序列中又连续出现三引号，则该三引号中的一个需要使用转义字符进行转义。

示例代码如下所示。

```
>>> str1 = ''                          # 创建名为 str1 的空字符串
>>> str1
''
>>> str2 = 'I LOVE CHINA'              # 创建名为 str2 的字符串
>>> str2
'I LOVE CHINA'
>>> str3 = " 天下兴亡，匹夫有责。"      # 创建名为 str3 的字符串
>>> str3
' 天下兴亡，匹夫有责。'
```

另外，英文三引号或 3 个英文双引号为定界符时可以换行创建字符串。示例代码如下所示。

```
>>> # 创建名为 str4 的字符串，字符串分两行输入
>>> str4 = ''' 绿水青山
就是金山银山 '''
>>> str4
' 绿水青山 \n 就是金山银山 '                # 字符串中的 \n 为转义字符，表示换行
>>> # 创建名为 str5 的字符串，字符串分两行输入
>>> str5 = """ 人民有信仰，
民族有希望，国家有力量。"""
>>> str5
' 人民有信仰，\n 民族有希望，国家有力量。'  # 字符串中的 \n 为转义字符，表示换行
```

2. 函数法

使用内置函数 str() 进行字符串的创建，基本语法格式如下所示。

```
字符串名 =str( 可迭代对象 )
```

该函数将可迭代对象转换为字符串，参数是可迭代对象如字符串、列表、元组、字典等。示例代码如下所示。

```
>>> str6 = str()                # 创建名为 str6 的空字符串
>>> str6
''
>>> str7 = str('CHINA')         # 创建名为 str7 的字符串
>>> str7
'CHINA'
>>> str8 = str([1, 2, 3])       # 创建名为 str8 的字符串
>>> str8
'[1, 2, 3]'
```

5.1.3　转义字符的使用


视频讲解

如果字符串本身就包含与定界符相同的元素，就会引发 SyntaxError 异常。示例代码如下所示。

```
# 字符串本身包含与定界符相同的元素，会引发 SyntaxError 异常
>>> strA = ' We're Chinese.'
SyntaxError: unterminated string literal.
```

为了避免上述的错误，可以采用以下两种解决办法。

（1）如果字符串中包含英文的单引号，那么整个字符串的边界符需要使用英文的双引号；如果字符串中包含英文的双引号，那么整个字符串的边界符需要使用英文的单引号。示例代码如下所示。

```
>>> strA =" We're Chinese. "                    # 字符串中的符号和边界符不同
>>> strA
" We're Chinese. "
```

（2）如果字符串中包含英文的单引号或双引号，可以在字符串中的引号之前添加反斜杠（\），对字符串中的引号进行转义，表示反斜杠后的引号是字符串中的一个普通字符，而不是字符串的定界符。示例代码如下所示。

```
>>> strA = ' We\'re Chinese.'                   # 字符串中的 \' 表示普通字符 '
>>> strA
" We're Chinese."
>>> strB = " 荀子曾言 :\" 学无止境，至死方休。\" "    # 字符串中的 \" 表示普通字符 "
>>> strB
' 荀子曾言 :" 学无止境，至死方休。" '
```

字符串中含有其他转义字符（如换行符或反斜杠符等）时处理方法的示例代码如下所示。

```
>>> print(" 人民有信仰 \n 民族有希望 \n 国家有力量 ")   # 字符串中含有换行符
人民有信仰
民族有希望
```

```
国家有力量
>>> print("C:\\Python\\Python310")                    # 字符串中含有反斜杠符
C:\Python\Python310
```

【例 5.1】编写程序，使用单引号或双引号作为字符串的定界符，通过 print() 语句实现以下内容的正确输出。

四个意识："政治意识\大局意识\核心意识\看齐意识"

四个自信：'道路自信 理论自信 制度自信 文化自信'

示例代码如下所示。

```
# 字符串定界符为''
print(' 四个意识："政治意识\\大局意识\\核心意识\\看齐意识" ')
# 字符串定界符为""
print(" 四个意识：\"政治意识\\大局意识\\核心意识\\看齐意识\" ")
# 字符串定界符为""
print(" 四个自信：'道路自信 理论自信 制度自信 文化自信' ")
# 字符串定界符为''
print(' 四个自信：\'道路自信 理论自信 制度自信 文化自信\' ')
```

运行结果如下所示。

```
四个意识："政治意识\大局意识\核心意识\看齐意识"
四个意识："政治意识\大局意识\核心意识\看齐意识"
四个自信：'道路自信 理论自信 制度自信 文化自信'
四个自信：'道路自信 理论自信 制度自信 文化自信'
```

5.1.4 字符串的截取

字符串的截取就是取出字符串中的子串。字符串的截取方法有两种：第一种是通过索引的方式进行单个字符的提取；第二种是通过切片的方式取出一段字符。

1. 字符串的索引

字符串是一个字符序列，可以通过索引的方式提取字符串中的单个字符。字符串的索引分为正向索引和反向索引。正向索引是把字符串中的字符从左向右进行编号，即第一个字符的索引号为 0，第二个字符的索引号为 1 等，以此类推。反向索引是把字符串中的字符从右向左进行编号，即最后一个字符的索引号为 -1，倒数第二个字符的索引号为 -2 等，以此类推。字符串索引的语法格式如下所示。

```
字符串对象名 [n]
```

其中，n 表示字符序列的索引号。正向索引时，索引范围是 [0，len(字符串对象名))，为一个左闭右开的区间；反向索引时，索引范围是 [- len(字符串对象名)，-1]，为一个闭区间。示例代码如下所示。

```
>>> strC = '仁义礼智信是指：仁爱 忠义 礼和 睿智 诚信'
>>> len(strC)                # 计算字符串长度，汉字、符号、空格都按 1 个字符长度计算
23
```

```
>>> strC[0]                    # 输出索引号为 0 的字符
'仁'
>>> strC[12]                   # 输出索引号为 12 的字符
'忠'
>>> strC[-2]                   # 输出索引号为 -2 的字符
'诚'
>>> strC[23]
# 索引号大于或等于字符串长度时，会引发 IndexError 异常
IndexError: string index out of range.
>>> strC['信']                 # 字符串索引号必须是整数，否则会引发 TypeError 异常
TypeError: string indices must be integers.
```

2. 字符串的切片

字符串是不可变序列，即字符串中的字符不允许修改。但是，可以通过切片操作的方式来提取字符串的子串。字符串切片操作的语法格式如下所示。

```
字符串对象名 [ 开始索引号 : 结束索引号 : 步长 ]
```

切片操作以冒号分隔开始索引号、结束索引号和步长，遵循“左闭右开”原则，即开始索引号所指定的字符会包含在切片内，而结束索引号所指定的字符不会包含在切片内。如果没有指定开始索引号，则默认为 0；如果没有指定结束索引号，则默认为字符串的长度；如果没有指定步长，则默认为 1。示例代码如下所示。

```
>>> strC = '仁义礼智信是指：仁爱 忠义 礼和 睿智 诚信'
>>> len(strC)                  # 计算字符串长度，汉字、符号、空格都按 1 个字符长度计算
23
>>> strC[0:22]                 # 返回索引号位于 [0,22) 区间内的字符
'仁义礼智信是指：仁爱 忠义 礼和 睿智 诚'
>>> strC[:]                    # 等价于 strC[::] 或 strC[0:23:1]，返回原字符串所有字符
'仁义礼智信是指：仁爱 忠义 礼和 睿智 诚信'
>>> strC[9:]                   # 等价于 strC[9:23] 或 strC[9:23:1]，返回从索引号 9 到
                               # 末尾的所有字符
'仁爱 忠义 礼和 睿智 诚信'
>>> strC[:5]                   # 等价于 strC[0:5] 或 strC[0:5:1]，返回索引号位于 [0,5)
                               # 区间内的字符
'仁义礼智信'
>>> strC[5:5]                  # 返回空字符串
''
```

需要注意的是，从左向右进行正向切片时，步长要求为正数；从右向左进行反向切片时，步长要求为负数，否则切片操作的结果为空字符串。示例代码如下所示。

```
>>> strC = '仁义礼智信是指：仁爱 忠义 礼和 睿智 诚信'
>>> strC[0:5:1]                # 正向切片，步长为正数，返回索引号位于 [0,5) 区间内的字符
'仁义礼智信'
>>> strC[0:5:-1]               # 正向切片，步长为负数，切片结果为空字符串
''
```

```
# 反向切片，步长为负数，返回索引号位于 [-1,-15) 区间内，间隔为 -3 的字符
>>> strC[-1:-15:-3]
'信智和义爱'
>>> strC[-1:-15:3]       # 反向切片，步长为正数，切片结果为空字符串
''
```

【例 5.2】对联，也称“楹联”“春联”等，是我国传统文化的瑰宝，2006 年国务院把楹联习俗列为第一批国家级非物质文化遗产名录。编写程序，判断输入的对联是否为回文联。例如，“雾锁山头山锁雾”“天连水尾水连天”是回文联。“金木水火土，生宇宙万物”“仁义礼智信，铸人世五常”不是回文联。

示例代码如下所示。

```
s = input("请输入一段话（输入 \"退出 \" 终止程序）：")
while s !="退出":
    if s == s[::-1]:
        print("这句话是回文联")
    else:
        print("这句话不是回文联")
    s = input("请输入一段话（输入 \"退出 \" 终止程序）：")
```

运行结果如下所示。

```
请输入一段话（输入 " 退出 " 终止程序）：雾锁山头山锁雾
这句话是回文联
请输入一段话（输入 " 退出 " 终止程序）：仁义礼智信，铸人世五常
这句话不是回文联
请输入一段话（输入 " 退出 " 终止程序）：退出
```

视频讲解

5.1.5 字符串常用方法

在 Python 中，有很多对字符串进行操作的方法，这些方法的名称可以使用内置函数 dir() 进行查看。示例代码如下所示。

```
>>> import string                # 导入 string 模块
>>> dir('srting')                # 该函数的参数为模块名 'srting'
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', 'capitalize', 'casefold', 'center',
'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans',
'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex',
```

```
    'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
    'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

在 Python 中，可以通过内置函数 help() 查询字符串对象方法的用法。示例代码如下所示。

```
>>> help(str.lower)              # 查询字符串对象 lower() 方法的用法
Help on method_descriptor:
lower(self, /)
    Return a copy of the string converted to lowercase.
```

此外，在 Python 中，查询字符串对象方法的用法，也可以通过一段程序代码实现，如例 5.3 所示。

【例 5.3】编写程序，输入要查询的字符串对象的方法名称，显示相对应的结果。

示例代码如下所示。

```
import builtins                  # 导入 builtins 模块
function_name = input("请输入要查询的字符串对象的方法名称：")
try:
    help(getattr(builtins.str, function_name))
# 调用内置函数 getattr() 获取字符串对象的方法名称 function_name
except AttributeError as e:
    print("未找到字符串对象的方法名称！")
```

运行结果如下所示。

```
请输入要查询的字符串对象的方法名称：lower
Help on method_descriptor:
lower(self, /)
    Return a copy of the string converted to lowercase.
```

再次运行程序，结果如下所示。

```
请输入要查询的字符串对象的方法名称：Upper
未找到字符串对象的方法名称！
```

常用的字符串方法有以下 5 种。

1. 字符串大小写转换

字符串大小写转换常用方法如表 5-1 所示。

表 5-1　字符串大小写转换常用方法

方　法	说　明
lower()	将字符串中所有字母转为小写字母
upper()	将字符串中所有字母转为大写字母
capitalize()	将字符串中的首字母大写，其余字母均为小写
title()	将字符串中所有单词的首字母大写，其余字母均为小写
swapcase()	将字符串中所有大写字母转为小写字母，所有小写字母转为大写字母

【例 5.4】编写程序，对字符串 " 天生我材必有用：All things in their being are good for something." 进行字符串大小写转换。

示例代码如下所示。

```
s = " 天生我材必有用：All things in their being are good for something. "
print(s.lower())          # 将字符串中所有字母转为小写字母
print(s.upper())          # 将字符串中所有字母转为大写字母
print(s.capitalize())     # 将字符串中的首字母大写，其余字母均为小写
print(s.title())          # 将字符串中所有单词的首字母大写，其余字母均为小写
print(s.swapcase())
# 将字符串中所有大写字母转为小写字母，所有小写字母转为大写字母
```

运行结果如下所示。

```
天生我材必有用：all things in their being are good for something.
天生我材必有用：ALL THINGS IN THEIR BEING ARE GOOD FOR SOMETHING.
天生我材必有用：all things in their being are good for something.
天生我材必有用：All Things In Their Being Are Good For Something.
天生我材必有用：aLL THINGS IN THEIR BEING ARE GOOD FOR SOMETHING.
```

2. 字符串格式输出

字符串格式输出常用方法如表 5-2 所示。

表 5-2　字符串格式输出常用方法

方　法	说　明
center()	将字符串按照给定的宽度居中输出，如果指定宽度小于字符串长度，则返回原字符串
ljust()	将字符串按照给定的宽度居左输出，如果指定宽度小于字符串长度，则返回原字符串
rjulst()	将字符串按照给定的宽度居右输出，如果指定宽度小于字符串长度，则返回原字符串
zfill()	将字符串按照给定的宽度输出，用“0”进行字符串的填充

【例 5.5】编写程序，对字符串“反诈骗电话：96110”进行字符串格式输出。

示例代码如下所示。

```
s1 = " 反诈骗电话 :96110"
print(s1.center(15,"*"))        # 居中输出，指定 "*" 进行填充
print(s1.ljust(15,"*"))         # 居左输出，指定 "*" 进行填充
print(s1.rjust(15,"*"))         # 居右输出，指定 "*" 进行填充
print(s1.zfill(15))             # 用 "0" 进行填充
```

运行结果如下所示。

```
** 反诈骗电话 :96110**
反诈骗电话 :96110****
**** 反诈骗电话 :96110
0000 反诈骗电话 :96110
```

3. 字符串搜索、定位和替换

字符串搜索、定位和替换常用方法如表 5-3 所示。

表 5-3　字符串搜索、定位和替换常用方法

方　法	说　明
count(sub[, start[, end]])	返回子字符串 sub 在字符串的 [start, end) 范围内出现的次数
find(sub[, start[, end]])	返回子字符串 sub 在字符串的 [start, end) 范围内首次出现的索引号，如果未检索到，返回 −1
rfind(sub[, start[, end]])	从右检测，返回子字符串 sub 在字符串的 [start, end) 范围内首次出现的索引号，如果未检索到，返回 −1
index(sub[, start[, end]])	返回子字符串 sub 在字符串的 [start, end) 范围内首次出现的索引号，如果未检索到，抛出异常
rindex(sub[, start[, end]])	从右检测，返回子字符串 sub 在字符串的 [start, end) 范围内首次出现的索引号，如果未检索到，抛出异常
replace(old, new[, count])	把字符串中的 old 字符替换为 new 字符，替换不超过 count 次
lstrip([chars])	截掉字符串左边的空格或指定字符 chars
rstrip([chars])	截掉字符串右边的空格或指定字符 chars
strip([chars])	截掉字符串左右两边的空格或指定字符 chars

【例 5.6】编写程序，对字符串“青年兴则国家兴，青年强则国家强。”进行字符串搜索、定位和替换。

示例代码如下所示。

```
s2 = "  青年兴则国家兴，青年强则国家强。  "
print(s2.count("青年"))      # 返回 "青年" 在字符串中出现的次数
print(s2.find("青年"))       # 返回 "青年" 在字符串中首次出现的索引号
print(s2.rfind("青年"))      # 从右检测，返回 "青年" 在字符串中首次出现的索引号
print(s2.replace("青年","少年"))     # 把字符串中的 "青年" 替换为 "少年"
print(s2.lstrip())                   # 截掉字符串左边的空格
print(s2.rstrip())                   # 截掉字符串右边的空格
print(s2.strip())                    # 截掉字符串左右两边的空格
```

运行结果如下所示。

```
2
2
10
  少年兴则国家兴，少年强则国家强。
青年兴则国家兴，青年强则国家强。
  青年兴则国家兴，青年强则国家强。
青年兴则国家兴，青年强则国家强。
```

4. 字符串联合和分隔

字符串联合和分隔常用方法如表 5-4 所示。

表 5-4　字符串联合和分隔常用方法

方　法	说　明
join()	用指定的字符作为连接符，将列表中多个字符串连接到一起
partition()	用指定的字符作为分隔符，从左向右将字符串分隔为两部分，返回 3 个元素的元组（头，分隔符，尾）
rpartition()	用指定的字符作为分隔符，从右向左将字符串分隔为两部分，返回 3 个元素的元组（头，分隔符，尾）
split()	用指定的字符作为分隔符，将字符串从左向右切割，返回列表，默认空格为分隔符，可以指定分隔次数
rsplit()	用指定的字符作为分隔符，将字符串从右向左切割，返回列表，默认空格为分隔符，可以指定分隔次数

【例 5.7】编写程序，进行字符串联合和分隔。

示例代码如下所示。

```
s3 = ["玉不琢","不成器","人不学","不知义"]
print("*".join(s3))                    # 用 "*" 进行列表中多个字符串的连接
s4 = "玉不琢-不成器-人不学-不知义"
# 从左向右，用 "-" 进行字符串的分隔，返回 3 个元素的元组
print(s4.partition("-"))
# 从右向左，用 "-" 进行字符串的分隔，返回 3 个元素的元组
print(s4.rpartition("-"))
# 从左向右，用 "-" 进行字符串的分隔，分隔两次，返回列表
print(s4.split("-",2))
# 从右向左，用 "-" 进行字符串的分隔，分隔 1 次，返回列表
print(s4.rsplit("-",1))
```

运行结果如下所示。

```
玉不琢*不成器*人不学*不知义
('玉不琢', '-', '不成器-人不学-不知义')
('玉不琢-不成器-人不学', '-', '不知义')
['玉不琢', '不成器', '人不学-不知义']
['玉不琢-不成器-人不学', '不知义']
```

5. 字符串条件判断

字符串条件判断常用方法如表 5-5 所示。

表 5-5　字符串条件判断常用方法

方　法	说　明
startswith(sub[, start[, end]])	判断字符串在指定 [start, end) 范围内，是否以指定字符串 sub 开始，是返回 True，否返回 False
endswith(sub[, start[, end]])	判断字符串在指定 [start, end) 范围内，是否以指定字符串 sub 结束，是返回 True，否返回 False
isalnum()	判断字符串至少有一个字符，并且都由字母或数字组成，是返回 True，否返回 False
isalpha()	判断字符串至少有一个字符，并且都由字母组成，是返回 True，否返回 False

续表

方　法	说　明
isdigit()	判断字符串至少有一个字符，并且都由数字组成，是返回 True，否返回 False
isidentifier()	判断字符串是否由有效 Python 标识符组成，是返回 True，否返回 False
islower()	判断字符串中的所有字母是否均为小写，是返回 True，否返回 False
isupper()	判断字符串中的所有字母是否均为大写，是返回 True，否返回 False
istitle()	判断字符串中每个单词是否首字母大写，其他字母小写，是返回 True，否返回 False
issapce()	判断字符串是否都由空格组成，是返回 True，否返回 False

【例 5.8】在 IDLE 交互式环境下，进行字符串条件判断。

示例代码如下所示。

```
>>> "2008年北京奥运会".startswith("2008"),"2008年北京奥运会".endswith("2008")
(True, False)
>>> "2008 Beijing Summer Olympics".isalnum(), "2008".isalpha(), "2008".
isdigit()
(False, False, True)
>>> "2008".isidentifier(),"_2008".isidentifier()
(False, True)
>>> "Beijing Summer Olympics".islower(), "Beijing Summer Olympics".isupper()
(False, False)
>>> "Beijing Summer Olympics".istitle(), "Beijing Summer Olympics".isspace()
(True, False)
```

5.2　正则表达式

正则表达式（Regular Expression，Regex）是一种强大的文本处理工具，它使用单个字符串来描述和匹配一系列符合某个特定语法规则的字符串。它通常被广泛应用于字符串搜索、替换以及复杂的数据验证等场景。

5.2.1　正则表达式概述

正则表达式是一个特殊的字符序列，它以灵活、简单的方法进行文本搜索和匹配、数据验证、数据提取、文本处理、语法分析和编译以及字符串操作，具体作用描述如下。

（1）文本搜索和匹配：可以用来搜索、匹配和替换特定模式的文本，例如查找所有符合特定格式的邮箱地址、电话号码等。

（2）数据验证：可以用来验证用户输入是否符合特定的格式要求，例如验证电子邮件地址、密码复杂度等。

（3）数据提取：可以从复杂的文本中提取出需要的信息，例如从网页源码中抽取出所有链接地址。

（4）文本处理：可以用来进行文本的分割、替换、删除等操作，例如删除所有的空格

或者特定标记。

（5）语法分析和编译：在编译器和解释器中，正则表达式可以用来识别和处理语法结构，例如在编程语言中的语法分析阶段，识别关键字、变量名等。

（6）字符串操作：可以用来处理和操作字符串，进行复杂的模式匹配和处理，例如对文本进行格式化、规范化等操作。字符串操作常用的功能有如下 4 种。

① 字符串提取：通过正则表达式来提取字符串中符合要求的文本。

② 字符串匹配：查看字符串是否符合正则表达式的语法，一般返回 True 或 False。

③ 字符串替换：查找字符串中符合正则表达式的文本，并且用指定的字符串进行替换。

④ 字符串分隔：使用正则表达式对字符串进行分隔。

5.2.2 正则表达式元字符

在 Python 中，使用正则表达式时，需要先导入 re 模块，re 模块提供了正则表达式操作所需要的功能。正则表达式通过普通字符和特殊字符（又称为元字符）构成一个规则（模式），使用这个规则对字符串进行过滤，从而得到想要的结果。

1. 字符类元字符

常用的字符类元字符如表 5-6 所示。

表 5-6 常用的字符类元字符

元 字 符	说 明
[ ]	匹配 [ ] 中所包含的任意一个字符
[^xyz]	反向字符集，匹配除 x、y、z 之外的任意字符
[a-z]	字符范围，匹配任意小写英文字母
[^a-z]	反向字符范围，匹配除小写英文字母之外的任意字符
[\u4e00-\u9fa5]	字符范围，匹配任意中文字符

【例 5.9】在 IDLE 交互式环境下，列举常用的字符类元字符。

示例代码如下所示。

```
>>> import re
>>> s = "中国:China 瓷器:china 中国人:chinese 唐人街:chinatown "
>>> re.findall(r"china",s)                    # 匹配所有的 china
['china', 'china']
>>> re.findall(r"[china]",s)                  # 匹配 c、h、i、n、a 中的任意一个字符
['h', 'i', 'n', 'a', 'c', 'h', 'i', 'n', 'a', 'c', 'h', 'i', 'n', 'c',
'h', 'i', 'n', 'a', 'n']
>>> re.findall(r"[A-Za-z]hina",s)             # 匹配大写字母或小写字母后面跟 hina
['China', 'china', 'china']
>>> re.findall(r"ch[in]a",s)                  # 匹配 chia 或者 chna
[ ]
>>> re.findall(r"chin[^a]",s)                 # 匹配 chin 后面不跟 a
['chine']
>>> re.findall(r"[[\u4e00-\u9fa5]",s)  # 匹配任意一个中文字符
['中', '国', '瓷', '器', '中', '国', '人', '唐', '人', '街']
```

2. 预定义字符类元字符

常用的预定义字符类元字符如表 5-7 所示。

表 5-7　常用的预定义字符类元字符

元　字　符	说　　明
.	匹配除换行符（\n）之外的任意单个字符
\	转义字符
\d	匹配任意一个数字，等价于 [0-9]
\D	匹配任意一个非数字，等价于 [^0-9]
\s	匹配任意一个空白字符，等价于 [\t\n\r\f\v]
\S	匹配任意一个非空白字符，与 \s 相反，等价于 [^\t\n\r\f\v]
\w	匹配字母、数字、下画线中的任意一个，等价于 [A-Za-z0-9_]
\W	匹配非字母、数字、下画线中的任意一个，与 \w 相反，等价于 [^A-Za-z0-9_]

【例 5.10】在 IDLE 交互式环境下，列举常用的预定义字符类元字符。

示例代码如下所示。

```
>>> import re
>>> s = "中国:China 瓷器:china 中国人:chinese 唐人街:chinatown "
>>> re.findall(r"chin.",s)          # 匹配 chin 后跟一个除换行符外的任意字符
['china', 'chine', 'china']
>>> re.findall(r".hin.",s)          # 匹配 hin 前后各跟一个除换行符外的任意字符
['China', 'china', 'chine', 'china']
>>> s1 = "华罗庚（1910.11.12 — 1985.6.12）是世界著名的数论家和组合学家。"
>>> re.findall(r"\d\d\d\d",s1)      # 匹配 4 位的任意数字
['1910', '1985']
>>> s2 = "1_one 2_two 3_three 4_four"
# 匹配一个数字后跟一个非空白字符，再跟一个字母或数字或下画线
>>> re.findall(r"\d\S\w",s2)
['1_o', '2_t', '3_t', '4_f']
```

3. 边界匹配符类元字符

常用的边界匹配符类元字符如表 5-8 所示。

表 5-8　常用的边界匹配符类元字符

元　字　符	说　　明
^	匹配字符串的头部
$	匹配字符串的尾部或换行符的前一个字符
\b	匹配一个单词的边界，即单词的头部或尾部
\B	匹配一个单词的非边界，即单词的非头部或非尾部，与 \b 相反

【例 5.11】在 IDLE 交互式环境下，列举常用的边界匹配符类元字符。

示例代码如下所示。

```
>>> import re
>>> s = "Cat, cat, catch that fat rat."
```

```
>>> re.findall(r"cat",s)        # 匹配所有的 cat
['cat', 'cat']
>>> re.findall(r"^cat",s)       # 匹配以 cat 开头的字符串
[]
>>> re.findall(r"at$",s)        # 匹配以 at 结尾的字符串
[]
>>> re.findall(r"\bcat",s)      # 匹配头部有边界的 cat
['cat', 'cat']
>>> re.findall(r"\Bat\B",s)     # 匹配头部和尾部都不是边界的 at
['at']
```

4. 重复限定符类元字符

常用的重复限定符类元字符如表 5-9 所示。

表 5-9　常用的重复限定符类元字符

元 字 符	说　明
*	匹配“*”前面的字符，0 次或多次
+	匹配“+”前面的字符，1 次或多次
?	匹配“?”前面的字符，0 次或 1 次
{}	匹配“{}”前面的字符，按“{}”中指定的次数匹配
|	匹配“|”之前或之后的字符
()	匹配“()”中的内容

【例 5.12】在 IDLE 交互式环境下，列举常用的重复限定符类元字符。

示例代码如下所示。

```
>>> import re
>>> s = "1 10 11 102 112 1122 1112"
>>> re.findall(r"111*",s)       # 匹配 "*" 前面的字符 1，0 次或多次
['11', '11', '11', '111']
>>> re.findall(r"112+",s)       # 匹配 "+" 前面的字符 2，1 次或多次
['112', '1122', '112']
>>> re.findall(r"11{2}",s)      # 匹配 "{ }" 前面的字符 1，2 次
['111']
>>> re.findall(r"12{2,}",s)     # 匹配 "{ }" 前面的字符 2，2 次或多次
['122']
>>> re.findall(r"102|112",s)    # 匹配 102 或者 112
['102', '112', '112', '112']
>>> re.findall(r"(112)+",s)     # 匹配 112，1 次或多次
['112', '112', '112']
```


视频讲解

5.2.3　正则表达式模块

Python 中，re 模块提供了众多可以实现正则表达式操作所需要的方法，这些方法名称可以通过内置函数 dir() 进行查看。

```
>>> import re
>>> dir(re)
['A', 'ASCII', 'DEBUG', 'DOTALL', 'I', 'IGNORECASE', 'L', 'LOCALE',
'M', 'MULTILINE', 'Match', 'Pattern', 'RegexFlag', 'S', 'Scanner', 'T',
'TEMPLATE', 'U', 'UNICODE', 'VERBOSE', 'X', '_MAXCACHE', '__all__',
'__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', '__version__', '_cache',
'_compile', '_compile_repl', '_expand', '_locale', '_pickle',
'_special_chars_map', '_subx', 'compile', 'copyreg', 'enum', 'error',
'escape', 'findall', 'finditer', 'fullmatch', 'functools', 'match',
'purge', 'search', 'split', 'sre_compile', 'sre_parse', 'sub', 'subn',
'template']
```

re 模块的常用方法如表 5-10 所示。

表 5-10　re 模块的常用方法

方　法	说　明
split()	将字符串用指定分隔符进行分隔，并返回分隔后的字符串列表
findall()	返回字符串中模式的所有匹配项的列表
sub()	用给定的字符串替换原字符串中的匹配项
escape()	对字符串中可能被解释为正则表达式的特殊字符进行转义
search()	在字符串中寻找正则表达式模式的第一次匹配
match()	在字符串开始位置进行正则表达式模式的匹配
compile()	创建模式的对象

1. split() 方法

split() 方法用指定的分隔符对字符串进行分隔操作，并返回分隔后的字符串列表。该方法的基本语法格式如下所示。

```
re.split(pattern,string[,flags])
```

其中，参数 pattern 表示正则表达式字符串；string 表示已知字符串或字符串对象；flags 表示匹配标志，用于控制正则表达式的匹配方式，如是否区分大小写、多行匹配等。匹配标志详细说明参见 5.2.4 节匹配选项编译标志。示例代码如下所示。

```
>>> import re
>>> s = "鼠、牛、虎、兔、龙、蛇、马、羊、猴、鸡、狗、猪"
>>> re.split("、",s)              # 以 "、" 为分隔符进行分隔
['鼠', '牛', '虎', '兔', '龙', '蛇', '马', '羊', '猴', '鸡', '狗', '猪']
>>> re.split("龙",s)              # 以 "龙" 为分隔符进行分隔
['鼠、牛、虎、兔、', '、蛇、马、羊、猴、鸡、狗、猪']
```

2. findall() 方法

findall() 方法用某个正则表达式模式对字符串进行匹配，按找到的顺序返回一个匹配列表。如果没有找到匹配的，则返回空列表。该方法的基本语法格式如下所示。

```
re.findall(pattern,string[,flags])
```

其中，各项参数含义同 split() 方法。示例代码如下所示。

```
>>> import re
>>> s = "一分耕耘，一分收获（No pains,no gains.)"
>>> re.findall(r"ai\w",s)    # 匹配 ai 后面跟一个字母或一个数字或一个下画线
['ain', 'ain']
>>> pattern = "[A-Z]\w"
>>> re.findall(pattern,s)    # 匹配大写字母后面跟一个字母或一个数字或一个下画线
['No']
```

3. sub() 方法

sub() 方法用给定的字符串替换原字符串中的匹配项。如果没有找到原字符串中的匹配项，则保留原字符串元素不变。该方法的基本语法格式如下所示。

```
re.sub(pattern,repl,string,count=0[,flags])
```

其中，参数 pattern 表示正则表达式字符串；repl 表示给定的字符串，也可以是一个函数；string 表示已知字符串或字符串对象；count 表示匹配后替换的最大次数，如果省略，则表示替换所有被匹配到的字符；flags 表示匹配标志，可以省略。示例代码如下所示。

```
>>> import re
>>> s = "您好！我的账户密码是：12345。"
>>> re.sub(r"[0-9]", "*",s)     # 将匹配到的每一个数字用 * 替换
'您好！我的账户密码是：*****。'
>>> re.sub(r"[0-9]+", "*",s)    # 将匹配到的多个数字用一个 * 替换
'您好！我的账户密码是：*。'
```

该示例中省略了 count 参数，结果为字符“*”替换了所有匹配到的数字。

4. escape() 方法

escape() 方法对字符串中可能被解释为正则表达式的特殊字符进行转义。该方法的基本语法格式如下所示。

```
re.escape(pattern)
```

其中，参数 pattern 表示含有特殊字符的字符串。示例代码如下所示。

```
>>> import re
>>> s = "中国政府网的网址为：https://www.gov.cn/"
# 字符串中的 . 容易被理解为正则表达式中的元字符
>>> re.escape(s)                # 对字符串中的 . 进行转义
'中国政府网的网址为：https://www\\.gov\\.cn/'
```

另外，需要注意，escape() 方法对特殊字符进行的转义可能会造成一些不符合预期的结果。

5. search() 方法

search() 方法利用正则表达式模式对给定的字符串进行匹配。如果匹配成功，则返回第一个匹配对象；否则返回 None。该方法的基本语法格式如下所示。

```
re.search(pattern,string[,flags])
```

其中，各项参数含义同 split() 方法。示例代码如下所示。

```
>>> import re
>>> s = "青年一代有理想、有本领、有担当，国家就有前途，民族就有希望。"
>>> re.search("青年", s)  # 在字符串 s 中对 "青年" 一词匹配，返回首次匹配成功的对象
<re.Match object; span=(0, 2), match='青年'>
>>> re.search("希望", s)  # 在字符串 s 中对 "希望" 一词匹配，返回首次匹配成功的对象
<re.Match object; span=(27, 29), match='希望'>
```

6. match() 方法

match() 方法从字符串的开始位置进行正则表达式模式匹配，如果匹配成功，则返回一个匹配对象，否则返回 None。该方法的基本语法格式如下所示。

```
re.match(pattern,string[,flags])
```

其中，各项参数含义同 split() 方法。示例代码如下所示。

```
>>> import re
>>> s = "青年一代有理想、有本领、有担当，国家就有前途，民族就有希望。"
>>> re.match("青年", s)
# 在字符串开始位置匹配 "青年" 一词，匹配成功返回匹配对象
<re.Match object; span=(0, 2), match='青年'>
>>> print(re.match("希望", s))
# 在字符串开始位置匹配 "希望" 一词，匹配失败返回 None
None
```

需要注意 match() 方法与 search() 方法的区别。match() 方法从字符串的开始位置匹配，如果字符串开始位置不符合正则表达式模式，则匹配失败，返回 None，即不再对后续字符匹配。而 search() 方法则对整个字符串进行扫描匹配，直到匹配成功。

7. compile() 方法

compile() 方法将一个字符串形式的正则表达式编译成一个正则表达式对象，以供 match()、search() 以及其他方法使用。该方法的基本语法格式如下所示。

```
p=re.compile(pattern[,flags])
```

其中，参数 pattern 是一个字符串形式的正则表达式；flags 表示匹配选项，默认为 0。等号左边的 p 是返回的正则表达式对象。示例代码如下所示。

```
>>> import re
>>> patter = re.compile(r"\d+")    # 通过 compile 创建正则表达式对象 patter
>>> s = "反诈骗电话 :96110"
>>> m= patter.findall(s)           # 利用 patter 对象提取字符串 s 中的数字
>>> print(m)
['96110']
```

【例 5.13】输入一个电话号码，该电话号码的格式为所在地电话区号 - 固定电话号码，其中所在地电话区号由 3 位或 4 位数字组成，且第一个数字为 0；固定电话号码由 7 位或 8 位数字组成。编写程序，使用正则表达式验证输入的电话号码是否为有效的电话号码。

示例代码如下所示。

```
import re
tel = input("请输入一个电话号码：")
# 通过 compile 创建模式对象 patter
patter = re.compile(r"(^0\d{3,4})-(\d{7,8}$)")
if patter.search(tel):
    print(tel,"是有效的电话号码。")
else:
    print(tel,"是无效的电话号码，请您重新核对。")
```

运行结果如下所示。

```
请输入一个电话号码：0951-12345678
0951-12345678 是有效的电话号码。
请输入一个电话号码：1234-12345678
1234-12345678 是无效的电话号码，请您重新核对。
```

5.2.4 匹配选项编译标志

正则表达式的匹配选项是通过编译标志对字符进行控制处理的，可以通过修改正则表达式的编译标志，进行某些特殊字符的处理。如匹配字符时是否区分大小写、多行匹配等。re 模块中常用的匹配选项编译标志如表 5-11 所示。

表 5-11 re 模块中常用的匹配选项编译标志

编译标志	描　述
re.A	部分转义符（如 \w、\b、\s、\d、\D 等）只能匹配 ASCII 码字符
re.I	匹配时不区分大小写
re.D	匹配所有字符，包括换行符等
re.M	多行匹配模式
re.S	匹配包括换行在内的所有字符

示例代码如下所示。

```
>>> import re
>>> re.findall('mother.country', 'mother\ncountry')     # 无匹配项
[]
>>> re.findall('mother.country', 'mother\ncountry',re.S) # re.S 匹配所有字符
['mother\ncountry']
>>> re.findall(r"New china","new China",re.I)           # re.I 不区分大小写
['new China']
```

小结

本章介绍了字符串和正则表达式的相关知识。字符串小节的相关知识，主要包括字符串的编码、创建、截取，转义字符的使用和字符串常用方法等。正则表达式小节的相

关知识，主要包括正则表达式的概述、元字符介绍、re 模块中常用方法及匹配选项编译标志等。

【思政元素融入】

在字符串编码中，从 ASCII 码到 Unicode 编码，需要了解不同编码方式的历史背景和相关文化，体现了对多元文化的尊重和包容。在转义字符的使用中，涉及对特殊字符的处理，体现了对数据安全和信息安全的重视。在字符串常用方法的使用中，需要对不同方法进行整合处理，体现了灵活应用所学知识解决实际问题的能力。在正则表达式元字符的使用中，需要按照特定的语法规则编写正则表达式，强调了正则表达式的规范性和严谨性，体现了对规范的尊重和遵守。这些内涵有利于学生综合素质、社会责任感和创新能力的培养。

习题

一、选择题

1. 以下关于字符串的描述正确的是（　　）。

A. 字符串中字符的长度为 1 或 2 或 3

B. 字符串中的字符可以进行数学运算，但进行数学运算的字符必须为数字

C. 在三引号字符串中可包含换行符、回车符等特殊字符

D. 字符串只可以使用一对英文单引号进行创建

2. 在 Python 中，使用函数方法创建空字符串的正确操作是（　　）。

A. str1 = ""　　B. str2 = "　"　　C. str3 = str( )　　D. str4 = str(　)

3. 在 Python 中，下列哪个选项是表示换行符的转义字符（　　）。

A. \n　　B. \t　　C. \r　　D. \v

4. 以下语句中，哪个选项是错误的字符串赋值方式，会出现 SyntaxError（　　）。

A. strA='满江红名句："莫等闲，白了少年头，空悲切。"'

B. strA="满江红名句："莫等闲，白了少年头，空悲切。""

C. strA="满江红名句：\"莫等闲，白了少年头，空悲切。\""

D. strA='满江红名句：\'莫等闲，白了少年头，空悲切。\''

5. 在 Python 中，以下程序的运行结果是（　　）。

```
tstr = '12345678'
print(tstr[1: -1:2])
```

A. 24　　B. 1357　　C. 246　　D. 2468

6. 在 Python 中，以下程序的运行结果是（　　）。

```
Str1="现在，青春是用来奋斗的；将来，青春是用来回忆的。"
print(Str1.replace("青春","岁月",1))
```

A. 现在，岁月是用来奋斗的；将来，岁月是用来回忆的。

B. 现在，青春是用来奋斗的；将来，岁月是用来回忆的。

C. 现在，青春是用来奋斗的；将来，青春是用来回忆的。

D. 现在，岁月是用来奋斗的；将来，青春是用来回忆的。

7. 在 Python 中，以下程序的运行结果是（　　）。

```
Str2 = "照镜子、正衣冠、洗洗澡、治治病"
split_Str2 = Str2.split("、", 1)
print(split_Str2)
```

A. ['照镜子','正衣冠','洗洗澡','治治病']
B. ['照镜子','正衣冠、洗洗澡、治治病']
C.('照镜子','正衣冠','洗洗澡','治治病')
D.('照镜子','正衣冠、洗洗澡、治治病')

8. 在 Python 中，执行以下语句，哪个选项所示的正则表达式可以匹配以"不忘初心"开头的字符串（　　）。

```
import re
s = "不忘初心，牢记使命。"
```

A. re.findall（r"不忘初心 $", s）　　B. re.findall（r"^ 不忘初心 ", s）
C. re.findall（r"\B 不忘初心 ", s）　　D. re.findall（r"\d 不忘初心 ", s）

9. 在 Python 中，以下哪个正则表达式可以匹配字符串中连续出现的数字"222"（　　）。

A. 2*　　B. 2+　　C. 2{3}　　D. 2{2,}

10. 在 Python 中，re 模块中的哪个方法用于将字符串用指定分隔符进行分隔，并返回分隔后的字符串列表（　　）。

A. findall()　　B. sub()　　C. split()　　D. search()

二、填空题

1. 在 Python 中，表示回车的转义字符是 _____。

2. 字符串 strA='打铁必须自身硬'，其中 strA[5] 表示的字符是 _____。

3. 字符串 strB='精准扶贫、精准脱贫'，其中 strB[7:9] 表示的字符是 _____。

4. 对字符串 strC="互联网+教育"进行 print(strC.center(9, "*")) 的操作，输出结果为 _____。

5. 对字符串 strD="绿水青山就是金山银山"进行 print(strD.rfind("山")) 的操作，输出结果为 _____。

6. 对字符串 strE="五讲-四美-三热爱"进行从右向左以连字符"-"为分隔的分割操作，切割一次后得到的元组为 _____。

7. 对字符串 strF="apple orange banana" 使用正则表达式 re.findall(r"\w\w\w\w", strF)，匹配的结果是 _____。

8. 对字符串 strG="abc123def456ghi" 进行 re.sub(r"\d+", "*", strG) 的操作，输出结果是 _____。

三、编程题

1. 输入一个包含数字的英文字符串，统计字母和数字出现的次数和频率。

2. 开发敏感词过滤程序，提示用户输入内容，如果用户输入的内容中包含“暴力”和“诈骗”，则将该内容用 ** 替换。例如，输入内容：“2023 年全国各地开展了护苗运动，提醒青少年上网要注意：涉及暴力、涉及诈骗的信息，不要轻易相信！”运行后输

出："2023 年全国各地开展了护苗运动，提醒青少年上网要注意：涉及 **、涉及 ** 的信息，不要轻易相信！"

3. 编写程序，提示用户输入一个包含数字的字符串，将字符串中的数字字符取出，生成一个新的字符串。

4. 编写程序，提示用户输入两个字符串，并取出两个字符串中的公共字符。

5. 编写程序，提示用户输入内容，统计字符串中单词的个数。

6. 输入一个手机号码，手机号码格式为：第 1 位是 1，第 2～11 位可以是 3、4、5、6、7、8、9，使用正则表达式判断手机号码是否有效。

7. 编写程序，输入一个字符串，该字符串为网址文本，使用正则表达式验证该网址是否有效。本题有效网址的格式为：^(http|https):// 匹配起始位置以 http:// 或 https:// 开头；[a-zA-Z0-9.-]+ 匹配由一个或多个字母、数字、点号或连字符构成的域名部分；\. 匹配分隔域名和顶级域名的一个点号；[a-zA-Z]{2，4} 匹配由 2～4 个字母构成的顶级域名；(/\S*)? 匹配可选的路径部分，包括一个斜杠后面跟随 0 个或多个非空白字符；$ 匹配字符串的结束位置。