第4章

字符串

拓展阅读

学习目标

- 掌握字符串的定义方式，能够根据需求准确定义字符串
- 掌握格式化字符串的方式，能够通过%、format()或 f-string 格式化字符串
- 掌握字符串的查找与替换，能够通过 find()和 replace()方法实现字符串的查找与替换操作
- 掌握字符串的分割与拼接，能够通过 split()和 join()方法或者运算符+实现字符串的分割与拼接操作
- 熟悉删除字符串的指定字符的方式，能够通过 strip()、lstrip()和 rstrip()方法删除字符串的指定字符
- 掌握字符串的大小写转换，能够实现字符串的大小写转换操作
- 熟悉字符串的对齐，能够通过 center()、ljust()、rjust()方法实现字符串的对齐

在计算机领域中，字符串是一种应用十分广泛的数据。它用于表示和处理文本类型的数据，比如账号、密码等。另外，Python 为字符串提供了丰富的功能，能够对字符串进行操作，包括字符串的查找与替换、字符串的分割与拼接、字符串大小写转换等。本章将对字符串的相关知识进行详细讲解。

4.1 字符串介绍

在 Python 中，字符串是由一系列字符组成的不可变序列，这些字符包括字母、数字、符号，以及其他如中文汉字或表情符号等的 Unicode 字符。Python 中支持使用单引号、双引号和三引号定义字符串，其中单引号和双引号通常用于定义单行字符串，三引号（包括三单引号和三双引号）通常用于定义多行字符串，示例代码如下：

```
print('千里之行，始于足下。')
print("千里之行，始于足下。")
print("""合抱之木，生于毫末；
九层之台，起于累土；
千里之行，始于足下。""")
```

运行代码，结果如下所示：

```
千里之行，始于足下。
千里之行，始于足下。
合抱之木，生于毫末；
九层之台，起于累土；
千里之行，始于足下。
```

引号除了可以定义字符串，还可以作为字符串的组成部分，例如，“let's learn Python”中包含一个单引号。此时若使用单引号进行定义，Python 解释器会将“let's learn Python”中的单引号与定义字符串的第一个单引号进行配对，认为字符串包含的内容至此结束，因此会出现语法错误，示例代码如下：

```
print('let's learn Python')
```

运行代码，结果如下所示：

```
File "E:\FastPrograms3\Chapter04\code01.py", line 1
    print('let's learn Python')
                          ^
SyntaxError: unterminated string literal (detected at line 1)
```

当遇到以上情景时，可以选择字符串本身不包含的双引号或三引号定义字符串。例如，将上述示例中定义字符串时使用的单引号分别修改为双引号或三引号，修改后的代码如下。

```
print("let's learn Python")
print("""let's learn Python""")
print('''let's learn Python''')
```

运行代码，结果如下所示：

```
let's learn Python
let's learn Python
let's learn Python
```

同理，若字符串中包含双引号，则可以使用单引号或三引号定义字符串；若字符串中包含三引号，则可以使用单引号或双引号定义字符串，以确保 Python 解释器可以按照预期对引号进行配对。

除此之外，还可以利用反斜线“\”对引号转义来实现以上功能。在字符串的引号前添加反斜线“\”，此时 Python 解释器会将反斜线“\”之后的引号视为一个普通字符，而非特殊符号。例如，使用单引号定义字符串'let's learn Python'，在该字符串中的单引号前面添加反斜线“\”，示例代码如下：

```
print('let\'s learn Python')
```

运行代码，结果如下所示：

```
let's learn Python
```

以上代码对字符串中的单引号进行转义，此方法同样适用于对字符串中的双引号或反斜线进行转义，示例代码如下：

```
print("How do you spell the word \"Python\"?")
print("E:\\Python\\new_features.txt")
```

运行代码，结果如下所示：

```
How do you spell the word "Python"?
E:\Python\new_features.txt
```

多学一招：转义字符

一些普通字符与反斜线组合后将失去其原有含义，产生新的含义。类似这样的由“\”组

合而成的、具有特殊意义的字符就是转义字符。转移字符通常用于表示一些无法显示的字符，例如制表符、回车符等。Python 中常用的转义字符如表 4-1 所示。

表 4-1　Python 中常用的转义字符

转义字符	功能说明
\b	退格符，用于删除光标前面的一个字符
\n	换行符，用于另起一行
\v	纵向制表符
\t	横向制表符
\r	回车符，用于将光标移至当前行的起始位置

为了帮助读者更好地理解转义字符的作用，接下来，通过示例演示转义字符，示例代码如下：

```
print('转义字符中:\n 表示换行;\r 表示回车;\b 表示退格')
```

运行代码，结果如下所示：

```
转义字符中:
表示回车表示退格
```

从上述结果可以看出，冒号后面的内容另起了一行，缺少了“表示换行;”这部分内容，“表示回车”与“表示退格”之间没有分号。

如果字符串中包含多个转义字符，但又不希望转义字符产生作用，此时可以使用原始字符串。在原始字符串中，任何字符都失去了其特殊含义，保持字面的含义。Python 中可以直接在字符串开始的引号之前添加前缀 r 或 R，使字符串成为原始字符串。示例代码如下：

```
print(r'转义字符中:\n 表示换行;\r 表示回车;\b 表示退格')
```

运行代码，结果如下所示：

```
转义字符中:\n 表示换行;\r 表示回车;\b 表示退格
```

从上述结果可以看出，字符串里面的内容按照原样输出，没有出现换行或者删除字符的效果。

4.2　格式化字符串

格式化字符串是指在字符串的指定位置暂时插入一些用于占位的格式符，这些用于占位的格式符在执行字符串时会被真实的值替换，从而动态地构建字符串的内容，以保持一致的格式。Python 中有 3 种格式化字符串的方式，分别是使用%格式化字符串、使用 format()方法格式化字符串和使用 f-string 格式化字符串。本节将针对格式化字符串的 3 种方式进行详细讲解。

4.2.1　使用%格式化字符串

字符串的格式化可以使用%实现，其语法格式如下：

```
format % values
```

以上语法格式中 format 表示一个字符串，该字符串中可以插入单个或多个为真实数据占位的格式符；values 表示单个或多个真实数据，多个真实数据以元组的形式进行存储；%代表执行格式化操作，即将 format 中的格式符替换为 values。

Python 中常见的格式符如表 4-2 所示。

表 4-2 Python 中常见的格式符

格式符	格式说明
%s	用于将对应的数据格式化为字符串
%d	用于将对应的数据格式化为有符号的十进制数
%o	用于将对应的数据格式化为有符号的八进制数
%x	用于将对应的数据格式化为有符号的十六进制数
%f	用于将对应的数据格式化为浮点数，可指定浮点数的精度，结果默认保留 6 位小数
%e	用于将对应的数据格式化为用科学记数法表示的浮点数，e 小写
%E	用于将对应的数据格式化为用科学记数法表示的浮点数，E 大写
%c	用于将对应的数据（整数或包含单个字符的字符串）格式化为字符

表 4-2 中罗列的格式符均由%和字符组成，其中%用于标识格式符的起始，它后面的字符表示真实数据被转换的类型。

接下来，使用%对字符串进行格式化操作，示例代码如下：

```
value = 10
format = '我今年%d 岁。'
print(format % value)
```

以上代码中首先定义了一个变量 value，它的值是十进制数 10，然后定义了一个字符串 format，该字符串中插入了一个用于格式化十进制数的格式符%d，最后使用%格式化字符串 format，将该字符串中的格式符%d 替换为 value 的值。

运行代码，结果如下所示：

```
我今年 10 岁。
```

需要注意的是，如果被替换数据的数据类型不符合格式符指定的数据类型，那么程序会出现类型异常的错误信息。例如，将上述示例代码中变量 value 的值修改为字符串，修改后的代码如下：

```
value = '10'
format = '我今年%d 岁。'              # 在字符串中插入用于格式化十进制数的格式符%d
print(format % value)                # 将%d 替换为变量 value 的值
```

运行代码，结果如下所示：

```
Traceback (most recent call last):
  File "E:\FastPrograms3\Chapter04\code02.py", line 3, in <module>
    print(format % value)        # 将%d 替换为变量 value 的值
          ~ ~ ~ ~ ~ ~ ~^~ ~ ~ ~ ~ ~
TypeError: %d format: a real number is required, not str
```

从上述结果的最后一条信息“TypeError: %d format: a real number is required, not str”可以看出，当程序使用格式符%d 格式化字符串时，需要的真实数据是一个整数，而不是字符串。

此外，还可以使用%对插入了多个格式符的字符串进行格式化操作，这时需要将多个真实数据放入元组中，按照顺序用元组中对应位置的真实数据依次替换格式符。注意，真实数据与格式符的数量和位置必须一致，以确保替换的准确性。示例代码如下：

```
name = '小明'
age = 27
address = '北京市昌平区'
print('我叫%s，今年%d 岁了，来自%s。' % (name, age, address))
```

运行代码，结果如下所示：

```
我叫小明，今年 27 岁了，来自北京市昌平区。
```

4.2.2　使用 format()方法格式化字符串

虽然使用%可以对字符串进行格式化，但是这种方式并不是很直观，开发人员一旦遗漏了替换数据或选择了不匹配的格式符，就会导致字符串格式化失败。为了能更便捷地格式化字符串，Python 为字符串提供了一个格式化方法 format()。format()方法的语法格式如下：

```
str.format(values)
```

以上语法格式中，str 表示需要被格式化的字符串，该字符串中可以插入单个或多个为真实数据占位的符号{}；values 表示单个或多个待替换的真实数据，多个数据以英文逗号分隔。

接下来，通过示例演示使用 format()方法格式化字符串，具体代码如下：

```
name = '小明'
string = '我叫{}'
print(string.format(name))
```

运行代码，结果如下所示：

```
我叫小明
```

从以上输出结果可以看出，原来字符串里面的符号{}被替换为变量 name 的值'小明'。由此可见，使用 format()方法格式化字符串时无须关注替换数据的类型。

字符串中也可以插入多个符号{}，此时使用 format()方法对字符串进行格式化，默认会按从左到右的顺序逐个将符号{}替换为真实的数据。示例代码如下：

```
name = '小明'
age = 27
# 字符串中插入了两个符号{}
string = '我叫{}，今年{}岁了'
# 使用 format()方法格式化字符串，并指定两个真实数据
print(string.format(name, age))
```

运行代码，结果如下所示：

```
我叫小明，今年 27 岁了
```

从以上输出结果可以看出，原来字符串中的第一个符号{}替换为变量 name 的值'小明'，第二个符号{}替换为变量 age 的值 27。

字符串的符号{}中可以明确地指定编号，这样在使用 format()方法对字符串进行格式化时，会按照编号从 values 中取出对应位置的真实数据来替换{}。编号是从 0 开始计数的，即第一个真实数据的编号为 0，第二个真实数据的编号为 1，以此类推。示例代码如下：

```
name = '小明'
age = 27
# 字符串中插入两个符号{}，并在{}中指定编号
string = '我叫{1}，今年{0}岁了'
# 使用 format()方法格式化字符串
print(string.format(age, name))
```

运行代码，结果如下所示：

```
我叫小明，今年 27 岁了
```

从以上输出结果可以看出，原来字符串中编号为 1 的{}替换为变量 name 的值'小明'，编

号为 0 的{}替换为变量 age 的值 27。

字符串的符号{}中可以明确指定变量名，这样在使用 format()方法对字符串进行格式化时，会根据变量名绑定的真实数据进行替换。示例代码如下：

```
name = '小明'
age = 27
# 字符串中插入两个符号{}，并在{}中指定变量名
string = '我叫{arg_one}，今年{arg_two}岁了'
# 使用 format()方法格式化字符串
print(string.format(arg_one=name, arg_two=age))
```

运行代码，结果如下所示：

```
我叫小明，今年 27 岁了
```

从以上输出结果可以看出，原来字符串中指定了变量名为 arg_one 的{}替换为变量 name 的值'小明'，指定了变量名为 arg_two 的{}替换为变量 age 的值 27。

另外，当使用 format()方法对字符串进行格式化时，如果待替换的真实数据为浮点数，则可以在字符串的符号{}中指定浮点数的精度，具体的语法格式为{:.*n*f}，其中 *n* 为小数点后的位数。示例代码如下：

```
value = 3.141592653589793
# 字符串中插入一个符号{}，并在{}中指定保留两位小数
string = 'π 的值为：{:.2f}'
result = string.format(value)
print(result)
```

运行代码，结果如下所示：

```
π 的值为：3.14
```

4.2.3 使用 f-string 格式化字符串

f-string 是一种更为简洁的格式化字符串的方式，它在形式上以 f 或 F 标识字符串，在字符串中的指定位置使用符号{}标明要替换的真实数据，符号{}中可以嵌入变量、表达式等。

例如，使用 f-string 格式化字符串，具体代码如下：

```
name = '小明'
age = 27
string = f'我叫{name}，今年{age}岁了'
print(string)
```

运行代码，结果如下所示：

```
我叫小明，今年 27 岁了
```

字符串的符号{}中还可以使用各种表达式，将表达式的执行结果作为要替换的真实数据，示例代码如下：

```
num1 = 9
num2 = 9
string = f"{num1}×{num2}={num1 * num2}"
print(string)
```

运行代码，结果如下所示：

```
9×9=81
```

此外，字符串的符号{}中还可以使用英文冒号为变量的值指定格式化的规则，例如指定

小数位数，示例代码如下：

```
value = 3.141592653589793
print(f'π 的值为：{value:.2f}.')
```

运行代码，结果如下所示：

```
π 的值为：3.14.
```

4.3　实训案例

4.3.1　地区时间格式转换器


地区时间格式转换器

地区时间格式转换器是用于将不同地区的时间表示格式进行转换的工具。根据用户选择的地区，该转换器将时间信息转换为对应地区的时间格式。本案例要求编写代码，制作一个地区时间格式转换器，具体要求如下。

（1）该转换器支持 8 个国家的时间格式转换，包括中国、美国、英国、澳大利亚、法国、德国、俄罗斯、加拿大。各国的标准时间格式如下。

- 中国：YYYY 年 MM 月 DD 日 HH:mm:ss。
- 美国：MM/DD/YYYY HH:mm:ss。
- 英国、澳大利亚、法国：DD/MM/YYYY HH:mm:ss。
- 德国、俄罗斯：DD.MM.YYYY HH:mm:ss。
- 加拿大：YYYY-MM-DD HH:mm:ss。

（2）用户需输入年、月、日、时、分、秒以及地区信息。

（3）该转换器根据用户选择的地区，将时间信息转换为对应地区的时间格式进行输出。

4.3.2　制作名片


制作名片

名片在当今社会交往活动中有着广泛的应用，用于介绍个人和公司信息，例如个人的姓名、职位、单位名称、电话号码、电子邮箱等信息，方便人们在交流后进行联系。本案例要求编写程序，实现根据用户输入的姓名、职位、电话号码、电子邮箱制作名片的功能。名片的样式具体如下。

```
============================
姓名：××××××××××
职位：××××××××××
电话号码：××××××××××
电子邮箱：××××××××××
============================
```

4.4　字符串的常见操作

字符串的操作在实际应用中是比较常见的。Python 内置了很多操作字符串的方法，使用这些方法可以轻松地实现字符串的查找、替换、拼接、大小写转换等操作。但需要注意的是，字符串一旦创建便不可修改；若对字符串进行修改，就会生成新的字符串。下面就结合 Python 内置方法对字符串的常见操作进行详细讲解。

4.4.1 字符串的查找与替换

为了维护良好的网络环境，一些平台会对用户发布的评论进行审核，一旦评论中包含敏感词，就需要将这些敏感词替换为*进行隐藏。评论通常是文本形式的字符串，查找与替换是实现文本过滤的基本操作，下面分别介绍如何实现字符串的查找与替换。

1. 字符串查找

Python 中提供了用于实现字符串查找操作的 find()方法。该方法可查找字符串中是否包含子串，若包含子串则返回子串首次出现的位置的索引，否则返回-1。

find()方法的语法格式如下所示：

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

以上语法格式中各参数的含义如下。

- sub：用于指定要查找的子串。
- start：用于指定查找的开始索引，默认值为 0。
- end：用于指定查找的结束索引，默认值为字符串的长度。

例如，查找'鱼'是否在字符串'与其临渊羡鱼，不如退而结网。'中，具体代码如下：

```
words = '与其临渊羡鱼，不如退而结网。'
result_one = words.find('鱼')          # 从整个字符串中查找子串'鱼'
print(result_one)
result_two = words.find('鱼', 6)       # 从索引 6 的位置开始查找子串'鱼'
print(result_two)
```

运行代码，结果如下所示：

```
5
-1
```

从第一个结果可以看出，成功找到了子串，子串出现的位置的索引是 5；从第二个结果可以看出，没有找到子串。

2. 字符串替换

Python 中提供了用于实现字符串替换操作的 replace()方法。该方法用于将当前字符串中的指定旧子串替换成新子串，可以指定替换次数，并返回替换后的新字符串。

replace()方法的语法格式如下所示：

```
replace(old, new[, count])
```

以上语法格式中各参数的含义如下。

- old：表示被替换的旧子串。
- new：表示用于替换旧子串的新子串。
- count：表示替换旧子串的次数，默认替换所有的旧子串。

下面演示如何使用 replace()方法实现字符串替换，示例代码如下：

```
string = "All things Are difficult before they Are easy."
new_string = string.replace("Are", "are")          # 不指定替换次数
print(new_string)
```

运行代码，结果如下所示：

```
All things are difficult before they are easy.
```

使用 replace()方法替换字符串的内容时指定替换次数，示例代码如下：

```
new_string = string.replace('Are', 'are', 1)    # 指定替换次数
print(new_string)
```

运行代码，结果如下所示：

```
All things are difficult before they Are easy.
```

从第一个结果可以看出，字符串中所有的 Are 替换成了 are；从第二个结果可以看出，字符串中的第一个 Are 替换成了 are，其余 Are 没有发生变化。

4.4.2　字符串的分割与拼接

字符串的分割与拼接功能是处理文本数据时常用的功能，下面分别介绍如何实现字符串的分割与拼接。

1. 字符串分割

split()方法用于根据指定分隔符对字符串进行分割，分割后返回一个列表，该列表中保存多个子串。

split()方法的语法格式如下所示：

```
split(sep=None, maxsplit=-1)
```

以上语法格式中各参数的含义如下。

- sep：表示分隔符，默认值为空格，也可被设置为其他字符，例如换行符（\n）、制表符（\t）等。
- maxsplit：分割次数，默认值为-1，表示不限制分割次数。

例如，分别以空格、字母 m 和字母 e 为分隔符对字符串"The more efforts you make, the more fortune you get."进行分割，示例代码如下：

```
string_example = "The more efforts you make, the more fortune you get."
print(string_example.split())         # 根据空格分割字符串
print(string_example.split('m'))      # 根据字母 m 分割字符串
print(string_example.split('e', 2))   # 根据字母 e 分割字符串，并且分割两次
```

运行代码，结果如下所示：

```
['The', 'more', 'efforts', 'you', 'make,', 'the', 'more', 'fortune',
'you', 'get.']
['The ', 'ore efforts you ', 'ake, the ', 'ore fortune you get.']
['Th', ' mor', ' efforts you make, the more fortune you get.']
```

2. 字符串拼接

join()方法用于将某个字符串作为连接符，通过连接符拼接可迭代对象的每个元素，并返回一个新的字符串。可迭代对象可以是字符串、列表、元组、集合、字典。

join()方法的语法格式如下：

```
join(iterable)
```

以上语法格式中，参数 iterable 表示可迭代对象。

例如，使用“*”拼接字符串'Python'中的各个字母，具体代码如下：

```
symbol = '*'
world = 'Python'
print(symbol.join(world))
```

运行代码，结果如下所示：

```
P*y*t*h*o*n
```

Python 中还可以使用运算符“+”将两个字符串拼接成一个字符串，示例代码如下：

```
start = 'Py'
```

```
end = 'thon'
print(start + end)
```

运行代码，结果如下所示：

```
Python
```

4.4.3 删除字符串的指定字符

字符串头部或尾部可能包含一些无用的字符，比如空格，当在程序中处理这种字符串时往往需要先删除这些无用的字符。Python 中的 strip()、lstip()和 rstrip()方法可以删除字符串头部或尾部的指定字符，如表 4-3 所示。

表 4-3 删除字符串的指定字符的方法

方法	功能说明
strip()	删除字符串头部和尾部指定的字符，默认删除空格
lstip()	删除字符串头部指定的字符，默认删除空格
rstrip()	删除字符串尾部指定的字符，默认删除空格

例如，分别删除字符串' Life is short, Use Python ! '头部和尾部的空格、头部的空格、尾部的空格，示例代码如下：

```
old_string = '  Life is short, Use Python !  '
strip_str = old_string.strip()                    # 删除字符串头部和尾部的空格
lstrip_str = old_string.lstrip()                  # 删除字符串头部的空格
rstrip_str = old_string.rstrip()                  # 删除字符串尾部的空格
print(f'strip()方法：{strip_str}】')
print(f'lstrip()方法：{lstrip_str}】')
print(f'rstrip()方法：{rstrip_str}】')
```

运行代码，结果如下所示：

```
strip()方法：Life is short, Use Python !】
lstrip()方法：Life is short, Use Python !】
rstrip()方法：  Life is short, Use Python !】
```

4.4.4 字符串大小写转换

在特定情况下，对于英文单词的大小写形式有一定的要求。例如，表示特殊简称时字母全大写，如 CBA；表示月份、周日、节假日时每个单词首字母大写，如 Monday。Python 中用于字符串大小写转换的方法有 upper()、lower()、capitalize()和 title()，如表 4-4 所示。

表 4-4 字符串大小写转换的方法

方法	功能说明
upper()	将字符串中的小写字母全部转换为大写字母
lower()	将字符串中的大写字母全部转换为小写字母
capitalize()	将字符串中的第一个字母转换为大写字母，其余字母转换为小写字母
title()	将字符串中每个单词的首字母转换为大写字母，其余字母转换为小写字母

例如，使用表 4-4 中提供的方法对字符串'hello woRld'进行大小写转换操作，示例代码如下：

```
old_string = 'hello woRld'
upper_str = old_string.upper()                    # 将字符串的字母转换为大写字母
lower_str = old_string.lower()                    # 将字符串的字母转换为小写字母
```

```
cap_str = old_string.capitalize()                    # 将字符串的首字母转换为大写字母
title_str = old_string.title()                       # 将每个单词的首字母转换为大写字母
print(f'upper()方法：{upper_str}')
print(f'lower()方法：{lower_str}')
print(f'capitalize()方法：{cap_str}')
print(f'title()方法：{title_str}')
```

运行代码，结果如下所示：

```
upper()方法：HELLO WORLD
lower()方法：hello world
capitalize()方法：Hello world
title()方法：Hello World
```

4.4.5　字符串对齐

在使用 Word 处理文档时可能需要对文档的对齐方式进行调整，如标题居中、左对齐、右对齐等。Python 中提供了设置字符串对齐方式的方法，分别是 center()、ljust()和 rjust()，如表 4-5 所示。

表 4-5　字符串对齐的方法

方法	功能说明
center(width[,fillchar])	使用 fillchar 填充字符串至指定长度，原字符串居中显示
ljust(width[,fillchar])	使用 fillchar 填充字符串至指定长度，原字符串左对齐显示
rjust(width[,fillchar])	使用 fillchar 填充字符串至指定长度，原字符串右对齐显示

表 4-5 罗列的方法中都有相同的参数，即 width 和 fillchar，其中参数 width 表示对齐后的字符串长度，如果参数 width 指定的长度小于或等于原字符串的长度，那么以上各方法会返回原字符串；参数 fillchar 表示填充的字符，默认值为空格。

接下来，使用表 4-5 中的方法对字符串'hello world'进行对齐操作，示例代码如下：

```
sentence = 'hello world'
center_str = sentence.center(13,'-')        # 字符串的长度为 13，居中显示，使用-填充
ljust_str = sentence.ljust(13, '*')         # 字符串的长度为 13，左对齐，使用*填充
rjust_st = sentence.rjust(13, '%')          # 字符串的长度为 13，右对齐，使用%填充
print(f"居中显示：{center_str}")
print(f"左对齐显示：{ljust_str}")
print(f"右对齐显示：{rjust_st}")
```

运行代码，结果如下所示：

```
居中显示：-hello world-
左对齐显示：hello world**
右对齐显示：%%hello world
```

4.5　实训案例


过滤不良词语

4.5.1　过滤不良词语

在互联网发展的初期，违法违规的乱象不断发生，严重危害了网络环境，侵犯了人民的正当权益。为了保障网络安全、维护国家和人民的利益、

推动信息化发展，大多数网络平台会采取过滤不良词语的策略来营造一个风清气正的网络环境。

在很多品牌的宣传中有些用词存在过度宣传的现象，没有客观依据证明，给消费者造成消费误导。这种过度宣传的用词属于不良词语。我们作为内容的生产者和发布者，都是网络语言生态的一分子，保持网络环境的健康纯洁，需要我们每个人从自身做起。

本案例要求对一段文本进行检测，一旦文本中出现不良词语“最优秀”，就将不良词语替换成“较优秀”，从而实现过滤不良词语的功能。文本内容如下。

我们拥有多年的品牌战略规划及标志设计、商标注册经验；专业提供公司标志设计与商标注册“一条龙”服务。我们拥有最优秀且具有远见卓识的设计师，因此我们的策略分析严谨，设计充满创意。我们有信心为您提供最优秀的品牌形象设计服务，为您的企业提升价值。

4.5.2 文字排版工具

文字排版工具

文字排版工具是一款强大的文章自动排版工具，会将文字按现代汉语习惯及发表出版要求进行规范编排。本案例要求编写代码，实现一个简易版的中文文字排版工具，具体要求如下。

（1）用户打开该工具时会看到欢迎提示信息，在该工具中可以输入要排版的文字。

（2）该工具一共有 5 个功能，分别是删除空格（编号 1）、英文标点替换（编号 2）、段落分割（编号 3）、字母大写（编号 4）和退出（编号 0），用户可以输入编号选择相应的功能。各功能的介绍如下。

- 删除空格：删除文字里面的任意位置的空格。
- 英文标点替换：将文字里面的英文逗号、句号、问号、感叹号、冒号分别替换为中文逗号、句号、问号、感叹号、冒号。
- 段落分割：将文字里面的\r\n 作为段落标记，使文字以多段内容进行显示。
- 字母大写：将文字里面的英文字母全部转换为大写英文字母。
- 退出：该工具输出感谢提示信息，并退出。注意，如果用户没有选择退出功能，则可以一直使用文字排版工具。

（3）该工具提供功能菜单，便于用户根据菜单的提示选择想要的功能。如果用户输入了错误的编号，则会输出无效选项提示信息。

4.6 本章小结

本章主要讲解了 Python 字符串的相关知识，包括字符串介绍、格式化字符串、字符串的常见操作，并结合实训案例演示了字符串的使用。通过对本章的学习，读者能够掌握字符串的使用。

4.7 习题

一、填空题

1. Python 中可以使用_____、双引号和三引号定义字符串。

2. Python 中使用_____方法可以删除字符串头部的空格。
3. Python 中可以使用_____运算符拼接字符串。
4. 使用 find()方法查找子串时，若没有查找到则返回_____。
5. f-string 在形式上以 f 或_____标识字符串。

二、判断题

1. 字符串中不可以包含特殊字符。（　　）
2. 使用单引号或双引号定义的字符串，通过 print()输出的结果都一致。（　　）
3. rjust()方法用于将字符串的字符以右对齐方式进行显示。（　　）
4. find()方法返回-1 说明子串在指定的字符串中。（　　）
5. strip()方法默认会删除字符串头部和尾部的空格。（　　）

三、选择题

1. Python 中使用（　　）转义字符。

   A. /　　B. \　　C. $　　D. %

2. 下列选项中，用于格式化字符串的是（　　）。

   A. %　　B. format()　　C. f-string　　D. 以上全部

3. 下列关于字符串的说法中，描述错误的是（　　）。

   A. 字符串创建后可以被修改

   B. Python 中可以使用三引号定义字符串

   C. 转义字符\n 表示换行符

   D. 格式符均由%和表示转换类型的字符组成

4. 下列方法中，可以将字符串中的字母全部转换为大写字母的是（　　）。

   A. upper()　　B. lower()　　C. title()　　D. capitalize()

5. 下列选项中，不属于字符串的是（　　）。

   A. "1"　　B. 'python'　　C. """^"""　　D. '1'.23

四、简答题

1. 请简述什么是字符串。
2. 请简述 Python 中格式化字符串的几种方式。
3. 请简述 Python 中用于实现字符串对齐的内置方法。

五、编程题

1. 编写程序，已知字符串 s = 'AbcDeFGhIJ'，请计算该字符串中小写字母的数量。
2. 编写程序，检查字符串" Life is short. I use python"中是否包含字符串"python"，若包含则将其替换为"Python"后输出新字符串，否则输出原字符串。