

第7章 字符串

第7章 字符串

- 在Python中，字符串属于不可变有序容器对象，使用单引号、双引号、三单引号或三双引号作为定界符，并且不同的定界符之间可以互相嵌套。

'abc'、'123'、'社会主义核心价值观'

"Python"

'''Tom said, "Let's go"'''

第7章 字符串

- 除了支持序列通用方法（包括双向索引、比较大小、计算长度、元素访问、切片、成员测试等操作）以外，字符串类型还支持一些特有的操作方法，例如字符串格式化、查找、替换、排版等等。
- 字符串属于不可变序列，不能直接对字符串对象进行元素增加、修改与删除等操作，字符串对象提供的所有方法都不会修改原字符串，切片操作也只能访问其中的元素，无法使用切片来修改字符串中的字符。

第7章 字符串

- `str`类型字符串可以通过`encode()`方法使用指定的字符串编码格式编码成为`bytes`类型的字节串对象，`bytes`类型的字节串对象可以通过`decode()`方法使用指定编码格式解码成为`str`字符串。

```
>>> type('中国')
<class 'str'>
>>> type('中国'.encode('gbk'))
<class 'bytes'>
>>> isinstance('中国', str)
True
>>> '中国'.encode()
b'\xe4\xb8\xad\xe5\x9b\xbd'
>>> '中国'.encode('gbk')
b'\xd6\xd0\xb9\xfa'
>>> bytes('董付国', 'gbk')
b'\xb6\xad\xb8\xb6\xb9\xfa'
>>> str(_, 'gbk')
'董付国'
```

7.1 字符串编码格式简介

- 最早的字符串编码是美国标准信息交换码ASCII，仅对10个数字、26个大写英文字母、26个小写英文字母及一些其他符号进行了编码。ASCII码采用1个字节来对字符进行编码，最多只能表示256个符号，表示能力非常有限。

7.1 字符串编码格式简介

- GB2312是我国制定的中文编码，使用1个字节表示英语，2个字节表示中文；GBK是GB2312的扩充，CP936是微软在GBK基础上开发的编码方式。GB2312、GBK和CP936都是使用2个字节表示中文。
- UTF-8对全世界所有国家需要用到的字符进行了编码，以1个字节表示英语字符(兼容ASCII)，以3个字节表示常见中文字符，还有些语言的符号使用2个字节（例如俄语和希腊语符号）或4个字节。

7.1 字符串编码格式简介

- 不同编码格式之间相差很大，采用不同的编码格式意味着不同的表示和存储形式，把同一字符存入文件时，写入的字节串内容可能会不同，在试图理解其内容时必须了解编码规则并进行正确的解码。如果解码方法不正确就无法还原信息，从这个角度来讲，字符串编码也具有加密的效果。

7.1 字符串编码格式简介

```
>>> '董付国'.encode('utf8')
b'\xe8\x91\xa3\xe4\xbb\x98\xe5\x9b\xbd'
>>> '董付国'.encode('cp936')
b'\xb6\xad\xb8\xb6\xb9\xfa'
>>> '董付国'.encode('cp936').decode('cp936')
'董付国'
```


7.1 字符串编码格式简介

```
>>> 'Python可以这样学'.encode('utf8').decode('cp936')
Traceback (most recent call last):
  File "<pyshell#63>", line 1, in <module>
    'Python可以这样学'.encode('utf8').decode('cp936')
UnicodeDecodeError: 'gbk' codec can't decode byte 0xaf in position 8:
illegal multibyte sequence
>>> 'Python程序设计开发宝典'.encode('cp936').decode('utf8')
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    'Python程序设计开发宝典'.encode('cp936').decode('utf8')
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb3 in position 6:
invalid start byte
>>> '测试'.encode().decode('gbk')      # 恰好能解码，但不是原来的字符串
'嫵嫵癡'
```

7.1 字符串编码格式简介

- Python 3.x完全支持中文字符，源程序默认使用UTF8编码格式，无论是一个数字、英文字母，还是一个汉字，在统计字符串长度时都按一个字符对待和处理。在Python 3.x中可以使用汉字作为标识符。

```
>>> import sys
```

```
>>> sys.getdefaultencoding()
```

```
'utf-8'
```

```
>>> s = '中国山东烟台ABCDE'
```

```
# 中文与英文字符同样对待，都算一个字符
```

```
>>> len(s)
```

```
11
```

```
>>> 姓名 = '张三'
```

```
# 使用中文作为变量名
```

```
>>> print(姓名)
```

```
# 输出变量的值
```

```
张三
```

7.2 转义字符与原始字符串

转义字符	含义	转义字符	含义
\b	退格，把光标移动到前一系列位置	\\	一个斜线\
\f	换页符	\'	单引号'
\n	换行符	\"	双引号"
\r	回车	\ooo	3位八进制数对应的字符
\t	水平制表符	\xhh	2位十六进制数对应的字符
\v	垂直制表符	\uhhhh	4位十六进制数表示的Unicode字符
\Uxxxxxxxx	8位十六进制数表示的Unicode字符		

7.2 转义字符与原始字符串

❖转义字符用法

```
>>> print('Hello\nWorld')
```

```
Hello
```

```
World
```

```
>>> print('\101')
```

```
A
```

```
>>> print('\x41')
```

```
A
```

```
>>> print('我是\u8463\u4ed8\u56fd')
```

```
我是董付国
```

```
>>> print('Python\tGo\tPHP\tJulia')
```

```
Python Go
```

```
PHP
```

```
Julia
```

\n表示换行符

三位八进制数对应的字符

两位十六进制数对应的字符

四位十六进制数表示Unicode字符

\t表示一个制表位

7.2 转义字符与原始字符串

- 为了避免对字符串中的转义字符进行转义，可以使用原始字符串，在字符串前面加上字母r或R表示原始字符串，其中的**所有字符都表示原始的字面含义而不会进行任何转义**，常用于文件路径、URL或正则表达式等场合。

```
>>> path = 'C:\Windows\notepad.exe'
```

```
>>> print(path)
```

字符\n被转义为换行符

```
C:\Windows
```

```
otepad.exe
```

```
>>> path = r'C:\Windows\notepad.exe' # 原始字符串，任何字符都不转义
```

```
>>> print(path)
```

```
C:\Windows\notepad.exe
```

7.2 转义字符与原始字符串

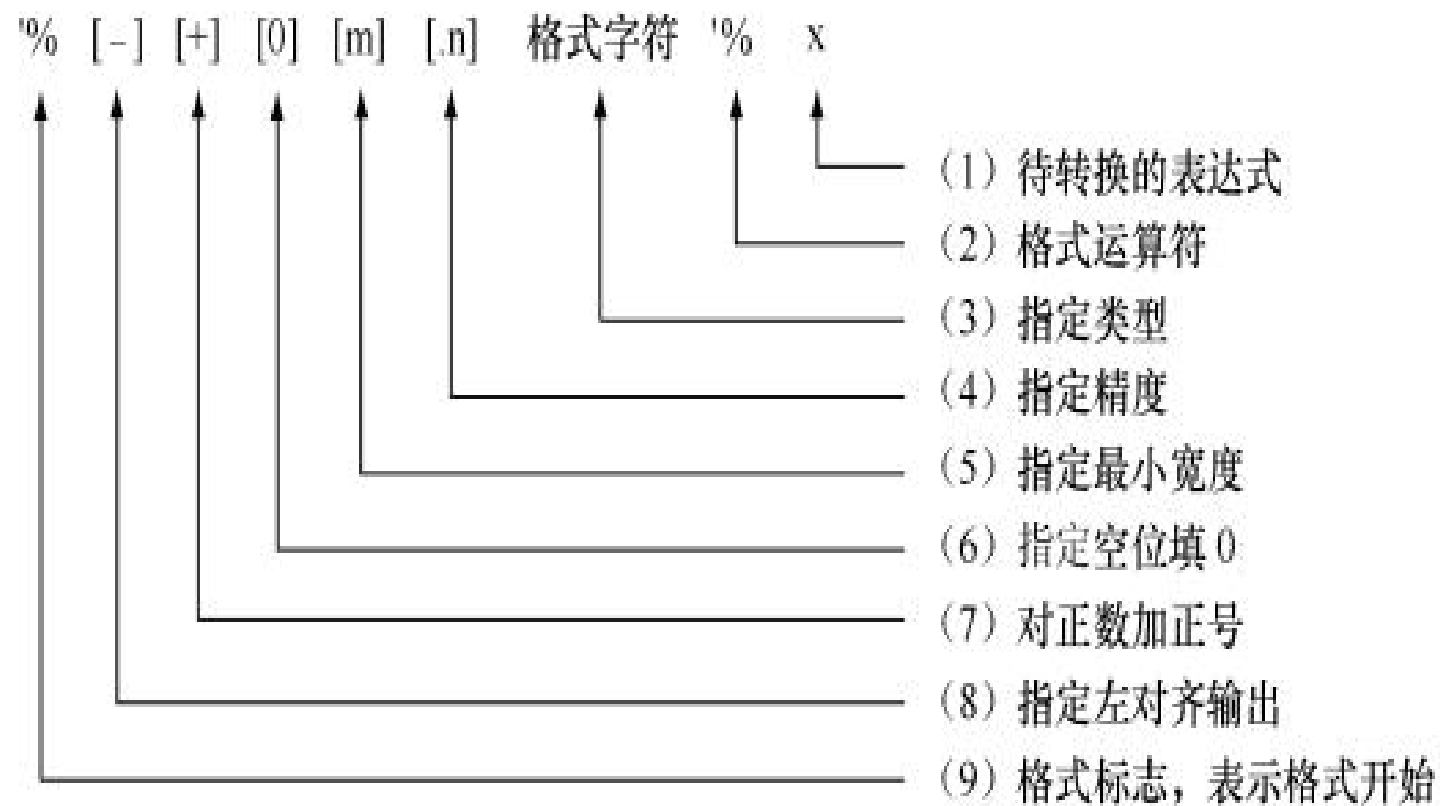
- f-字符串和原始字符串可以同时使用，在字符串定界符前面可以同时加字母r和f，位置和大小写没有限制。

```
>>> directory = 'child'
>>> filename = 'test.txt'
>>> fr'{directory}\{filename}'
'child\\test.txt'
>>> print(fr'{directory}\{filename}')
child\test.txt
```

7.3 字符串格式化

- 7.3.1 使用%运算符进行格式化
- 7.3.2 使用format方法进行格式化
- 7.3.3 格式化的字符串常量

7.3.1 使用%运算符进行格式化



7.3.1 使用%运算符进行格式化

- 常用格式字符

格式字符	说明
%s	字符串（采用 <code>str()</code> 的显示）
%r	字符串（采用 <code>repr()</code> 的显示）
%c	单个字符
%d	十进制整数
%i	十进制整数
%o	八进制整数
%x	十六进制整数
%e	指数（基底写为e）
%E	指数（基底写为E）
%f、%F	浮点数
%g	指数(e)或浮点数（根据长度决定显示方法）
%G	指数(E)或浮点数（根据长度决定显示方法）
%%	一个字符“%”

7.3.1 使用%运算符进行格式化

```
>>> x = 1235
>>> so = '%o' % x
>>> so
"2323"
>>> sh = '%x' % x
>>> sh
"4d3"
>>> se = '%e' % x
>>> se
"1.235000e+03"
>>> '%s' % 65
"65"
>>> '%s' % 65333
"65333"
>>> '%d' % '555'
TypeError: %d format: a number is required, not str
```

7.3.2 使用format()方法进行格式化

```
>>> 1/3
```

```
0.3333333333333333
```

```
>>> '{0:.3f}'.format(1/3)
```

```
'0.333'
```

```
>>> '{0:%}'.format(3.5)
```

格式化为百分数

```
'350.000000%'
```

```
>>> '{0:,},{0:_x}'.format(1000000)
```

Python 3.6.0新增支持下划线千分符

```
'1,000,000,f_4240'
```

```
>>> '{0:_},{0:,x}'.format(1000000)
```

格式化为十六进制数时不支持逗号千分符

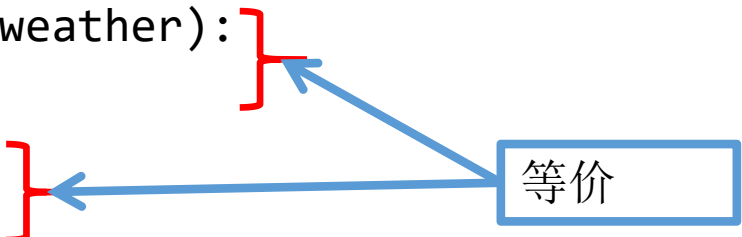
```
ValueError: Cannot specify ',' with 'x'.
```

7.3.2 使用format()方法进行格式化

```
>>> print('The number {0:,.} in hex is: {0:#x}, the number {1} in oct is {1:#o}'.format(5555, 55))
The number 5,555 in hex is: 0x15b3, the number 55 in oct is 0o67
>>> print('The number {1:,.} in hex is: {1:#x}, the number {0} in oct is {0:o}'.format(5555, 55))
The number 55 in hex is: 0x37, the number 5555 in oct is 12663
>>> print('my name is {name}, my age is {age}, and my QQ is {qq}'.format(name='Dong Fuguo',
    age=40, qq='30646****'))
my name is Dong Fuguo, my age is 40, and my QQ is 30646****
>>> position = (5, 8, 13)
>>> print('X:{0[0]};Y:{0[1]};Z:{0[2]}'.format(position))
X:5;Y:8;Z:13
>>> '{0[a]}'.format({'a':97, 'b':98})
'97'
>>> '{d[a]}'.format(d={'a':97, 'b':98})
'97'
```

7.3.2 使用format()方法进行格式化

```
weather = [('Monday','rainy'), ('Tuesday','sunny'),  
           ('Wednesday','sunny'), ('Thursday','rainy'),  
           ('Friday','cloudy')]  
formatter = "Weather of '{0[0]}' is '{0[1]}'".format  
for item in map(formatter,weather):  
    print(item)  
for item in weather:  
    print(formatter(item))
```



运行结果:

```
Weather of 'Monday' is 'rainy'  
Weather of 'Tuesday' is 'sunny'  
Weather of 'Wednesday' is 'sunny'  
Weather of 'Thursday' is 'rainy'  
Weather of 'Friday' is 'cloudy'
```

7.3.2 使用format()方法进行格式化

```
>>> '{0:=<8d},{0:#^8d},{0:->8d}'.format(666)    # 指定填充符、对齐方式、宽度  
'666=====,##666###,-----666'
```

7.3.3 格式化的字符串常量

- 从Python 3.6.x开始支持一种新的字符串格式化方式，官方叫做Formatted String Literals，在字符串前加字母f，简称f-字符串，功能与内置函数format()、字符串对象的format()方法类似。

```
>>> name = 'Dong'
>>> age = 39
>>> f'My name is {name}, and I am {age} years old.'
'My name is Dong, and I am 39 years old.'
>>> f'{10/4:.3}'                                     # 最多保留3位有效数字
'2.5'
>>> f'{10/3:.3}'
'3.33'
>>> f'{1000/3:.3}'
'3.33e+02'
```

7.3.3 格式化的字符串常量

```
>>> width = 10
>>> precision = 4
>>> value = 11/3
>>> f'result:{value:{width}.{precision}}'
'result:      3.667'
>>> f'result:{value:{width}.{precision}f}'
'result:      3.6667'
>>> f'result:{1111/3:{width}.{precision}}'
'result:      370.3'
>>> f'result:{1111/3:{width}.{precision}f}'
'result:  370.3333'
```

最多保留4位有效数字

恰好保留4位小数

7.3.3 格式化的字符串常量

```
>>> from datetime import date
>>> today = date.today()
>>> f'{today.year}年{today.month}月{today.day}日'
'2021年10月30日'
>>> for i, v in enumerate([1, 2, 3]):
    print(f'{i}:{v}')
```

0:1

1:2

2:3

```
>>> h, w = 3, 4
```

```
>>> f'{h*w=}'
```

'h*w=12'

Python 3.8以及更高版本支持

7.4 字符串常用操作

- Python字符串对象提供了大量方法用于字符串的切分、连接、替换和排版等操作，另外还有大量运算符和内置函数、标准库函数、扩展库函数也支持对字符串的操作。
- 字符串对象是不可变的，字符串对象提供的涉及到字符串“修改”的方法都是返回修改后的新字符串，并不对原始字符串做任何修改。

7.4 字符串常用操作

方法	功能描述
<code>capitalize()</code>	返回新字符串，第一个英文字母大写，其余小写
<code>center(width, fillchar=' ', /)</code> <code>ljust(width, fillchar=' ', /)</code> <code>rjust(width, fillchar=' ', /)</code>	返回指定长度的新字符串，当前字符串在新字符串中居中/居左/居右，如果参数 <code>width</code> 指定的新字符串长度大于当前字符串长度就在两侧/右侧/左侧使用参数 <code>fillchar</code> 指定的字符（默认为空格）进行填充；如果参数 <code>width</code> 指定的长度小于或等于当前字符串的长度，直接返回当前字符串，不会进行截断
<code>count(sub[, start[, end]])</code>	返回字符串 <code>sub</code> 在当前字符串下标范围 <code>[start,end)</code> 内不重叠出现的次数，参数 <code>start</code> 默认值为0，参数 <code>end</code> 默认值为字符串长度。例如， <code>'abababab'.count('aba')</code> 的值为2
<code>encode(encoding='utf-8', errors='strict')</code>	返回当前字符串使用参数 <code>encoding</code> 指定的编码格式编码后的字节串。对于非ASCII字符，UTF-8编码得到的字节串较长，网络传输时占用带宽较大，保存为文件时占用空间较大；GBK编码得到的字节串更短一些，但有些字符不在GBK字符集中，无法使用GBK编码
<code>endswith(suffix[, start[, end]])</code> <code>startswith(prefix[, start[, end]])</code>	如果当前字符串下标范围 <code>[start,end)</code> 的子串以某个字符串 <code>suffix/prefix</code> 或元组 <code>suffix/prefix</code> 指定的几个字符串之一结束/开始则返回True，否则返回False

7.4 字符串常用操作

<code>expandtabs(tabsize=8)</code>	返回新字符串，所有 <code>tab</code> 键都替换为指定数量的空格。 例如， <code>'abcd\te'.expandtabs()</code> 返回 <code>'abcd e'</code> ， <code>'abcdef\te'.expandtabs()</code> 返回 <code>'abcdef e'</code>
<code>find(sub[, start[, end]])</code> <code>rfind(sub[, start[, end]])</code>	返回字符串 <code>sub</code> 在当前字符串下标范围 <code>[start,end)</code> 内出现的最小/最大下标，不存在时返回-1
<code>format(*args, **kwargs)</code>	返回对当前字符串进行格式化（格式化是指把字符串中大括号以及内部变量或编号指定的占位符替换为实际值并以指定的格式呈现）后的新字符串，其中 <code>args</code> 表示位置参数， <code>kwargs</code> 表示关键参数，见5.2节
<code>index(sub[, start[, end]])</code> <code>rindex(sub[, start[, end]])</code>	返回字符串 <code>sub</code> 在当前字符串下标范围 <code>[start,end)</code> 内出现的最小/最大下标，不存在时抛出 <code>ValueError</code> 异常并提示子串不存在
<code>isalnum()</code> 、 <code>isalpha()</code> 、 <code>isascii()</code> 、 <code>isdecimal()</code> 、 <code>isdigit()</code> 、 <code>isidentifier()</code> 、 <code>islower()</code> 、 <code>isnumeric()</code> 、 <code>isprintable()</code> 、 <code>isspace()</code> 、 <code>istitle()</code> 、 <code>isupper()</code>	测试字符串中字符的类型

7.4 字符串常用操作

<code>join(iterable, /)</code>	使用当前字符串作为连接符把参数 <code>iterable</code> 中的所有字符串连接成为一个长字符串并返回连接之后的长字符串，要求参数 <code>iterable</code> 指定的可迭代对象中所有元素全部为字符串
<code>lower()</code> <code>upper()</code>	返回当前字符串中所有字母都变为小写/大写之后的新字符串，非字母字符保持不变
<code>lstrip(chars=None, /)</code> <code>rstrip(chars=None, /)</code> <code>strip(chars=None, /)</code>	返回当前字符串删除左侧/右侧/两侧的空白字符或参数 <code>chars</code> 中所有字符之后的新字符串
<code>maketrans(...)</code>	根据参数给定的字典或者两个等长字符串对应位置的字符，构造并返回字符映射表（形式上是字典，“键”和“值”都是字符的Unicode编码），如果指定了第三个参数（必须为字符串）则该参数中所有字符都被映射为空值 <code>None</code> 。该方法是字符串类 <code>str</code> 的静态方法，可以通过任意字符串进行调用，也可以直接通过字符串类 <code>str</code> 进行调用
<code>partition(sep, /)</code> <code>rpartition(sep, /)</code>	使用参数 <code>sep</code> 指定字符串的第一次/最后一次出现作为分隔符，把当前字符串分隔为3个子串，返回包含3个子串的元组。如果指定的子串不存在，返回当前字符串和两个空串组成的元组

7.4 字符串常用操作

<code>removeprefix(prefix, /)</code> <code>removesuffix(suffix, /)</code>	返回删除前缀/后缀之后的新字符串
<code>replace(old, new, count=-1, /)</code>	返回当前字符串中所有子串 <code>old</code> 都被替换为子串 <code>new</code> 之后的新字符串，参数 <code>count</code> 用来指定最大替换次数，默认值 <code>-1</code> 表示全部替换
<code>rsplit(sep=None, maxsplit=-1)</code> <code>split(sep=None, maxsplit=-1)</code>	使用参数 <code>sep</code> 指定的字符串对当前字符串从后向前/从前向后进行切分，返回包含切分后所有子串的列表。参数 <code>sep=None</code> 表示使用所有空白字符作为分隔符并丢弃切分结果中的所有空字符串，参数 <code>maxsplit</code> 表示最大切分次数，默认值 <code>-1</code> 表示没有限制
<code>splitlines(keepends=False)</code>	使用换行符切分字符串，返回列表
<code>swapcase()</code>	交换英文字母大小写，返回新字符串
<code>title()</code>	返回新字符串，每个单词的首字母大写，其余字母小写
<code>translate(table, /)</code>	根据参数 <code>table</code> 指定的映射表对当前字符串中的字符进行替换并返回替换后的新字符串，不影响原字符串，参数 <code>table</code> 一般为字符串方法 <code>maketrans()</code> 创建的映射表，其中映射为空值 <code>None</code> 的字符将会被删除而不出现在新字符串中

7.4.1 find()、rfind()、index()、rindex()、count()

- `find(sub[, start[, end]])`和`rfind(sub[, start[, end]])`方法分别用来查找另一个字符串在当前字符串指定范围（默认是整个字符串）中首次和最后一次出现的位置，如果不存在则返回-1；
- `index(sub[, start[, end]])`和`rindex(sub[, start[, end]])`方法用来返回另一个字符串在当前字符串指定范围中首次和最后一次出现的位置，如果不存在则抛出异常；
- `count(sub[, start[, end]])`方法用来返回另一个字符串在当前字符串中出现的次数。

7.4.1 find()、rfind()、index()、rindex()、count()

```
>>> s = 'apple,peach,banana,peach,pear'
>>> s.find('peach')
6
>>> s.find('peach', 7)
19
>>> s.find('peach', 7, 20)
-1
>>> s.rfind('p')
25
>>> s.index('p')
1
>>> s.index('pe')
6
```

```
>>> s.index('pear')
25
>>> s.index('ppp')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in
<module>
    s.index('ppp')
ValueError: substring not found
>>> s.count('p')
5
>>> s.count('pp')
1
>>> s.count('ppp')
0
```


7.4.2 split()、rsplit()

- `split(sep=None, maxsplit=-1)` 和 `rsplit(sep=None, maxsplit=-1)` 方法分别用来以参数字符串为分隔符，把当前字符串从左往右或从右往左分隔成多个字符串，返回包含分隔结果的列表。

7.4.2 split()、rsplit()

```
>>> s = 'apple,peach,banana,pear'
>>> s.split(',')
['apple', 'peach', 'banana', 'pear']
>>> s = '2022-7-31'
>>> t = s.split('-')
>>> print(t)
['2022', '7', '31']
>>> print(list(map(int, t)))
[2022, 7, 31]
```



分隔符

7.4.2 split()、rsplit()

- `split()` 和 `rsplit()` 方法允许通过参数 `maxsplit` 指定最大分割次数，该参数默认值 `-1` 表示没有限制。

```
>>> s = '\n\nhello\t\t world \n\n\n My name is Dong '
>>> s.split(None, 1)          # 第一个参数None表示使用任意空白字符做分隔符
['hello', 'world \n\n\n My name is Dong ']
```

```
>>> s.rsplit(None, 2)
['\n\nhello\t\t world \n\n\n My name', 'is', 'Dong']
```

```
>>> s.split(maxsplit=6)       # 使用关键参数直接指定第二个参数
                                # 这时第一个参数使用默认值None
['hello', 'world', 'My', 'name', 'is', 'Dong']
```

```
>>> s.split(maxsplit=100)     # 最大分隔次数大于可分隔次数时，相当于-1
['hello', 'world', 'My', 'name', 'is', 'Dong']
```

7.4.2 split()、rsplit()

- 对于split()和rsplit()方法，如果**不指定分隔符**，则字符串中的任何空白字符（空格、换行符、制表符等）都将被认为是分隔符，并删除分割结果中的所有空字符串。

```
>>> s = 'hello world \n\n My name is Dong '
>>> s.split()
['hello', 'world', 'My', 'name', 'is', 'Dong']
>>> s = '\n\nhello world \n\n\n My name is Dong '
>>> s.split()
['hello', 'world', 'My', 'name', 'is', 'Dong']
>>> s = '\n\nhello\t\t world \n\n\n My name\t is Dong '
>>> s.split()
['hello', 'world', 'My', 'name', 'is', 'Dong']
```

7.4.2 split()、rsplit()

◆然而，明确传递参数指定split()使用的分隔符时，情况是不一样的。

```
>>> 'a,,,bb,,,ccc'.split(',')          # 每个逗号都被作为独立的分隔符
                                         # 相邻逗号直接会得到一个空字符串
['a', '', '', 'bb', '', 'ccc']

>>> 'a\t\t\tbb\t\tccc'.split('\t')      # 每个制表符都被作为独立的分隔符
                                         # 相邻制表符之间会得到一个空字符串
['a', '', '', 'bb', '', 'ccc']

>>> 'a\t\t\tbb\t\tccc'.split()           # 连续多个制表符被作为一个分隔符
['a', 'bb', 'ccc']
```

7.4.3 join()

- 字符串方法 `join(iterable, /)` 用于连接可迭代对象中的字符串，以当前字符串作为连接符，返回连接后的新字符串。

连接符

```
>>> li = ['apple', 'peach', 'banana', 'pear']
```

```
>>> ','.join(li)
```

```
'apple,peach,banana,pear'
```

```
>>> '.'.join(li)
```

```
'apple.peach.banana.pear'
```

```
>>> '::'.join(li)
```

```
'apple::peach::banana::pear'
```

7.4.3 join()

- 应用：使用split()和join()方法删除字符串中多余的空白字符，连续多个空白字符只保留一个。

```
def equavilent(s1, s2):          # 判断两个字符串在Python意义上是否等价
    if s1 == s2:
        return True
    elif ' '.join(s1.split()) == ' '.join(s2.split()):
        return True
    elif ''.join(s1.split()) == ''.join(s2.split()):
        return True
    else:
        return False
```

```
print(equavilent('pip list', 'pip    list'))
print(equavilent('[1,2, 3, 4]', '[1, 2, 3, 4]'))
```

True
True

7.4.4 lower()、upper()、capitalize()、title()、swapcase()

■ lower()、upper()、capitalize()、title()、swapcase()

```
>>> s = 'What is Your Name?'
>>> s.lower()                # 返回小写字符串
'what is your name?'
>>> s.upper()                # 返回大写字符串
'WHAT IS YOUR NAME?'
>>> s.capitalize()          # 字符串首字符大写
'What is your name?'
>>> s.title()                # 每个单词的首字母大写
'What Is Your Name?'
>>> s.swapcase()             # 大小写互换
'wHAT IS yOUR nAME?'
```


7.4.5 replace()、maketrans()、translate()

- 字符串对象的`replace(old, new, count=-1, /)`方法把指定子字符串的所有出现都替换为另一个字符串，类似于Word中的“全部替换”功能，返回替换后的新字符串。必要时可以通过参数`count`指定替换次数。

```
>>> s = '中国，中国'
>>> s2 = s.replace('中国', '中华人民共和国') # 两个参数都作为一个整体
>>> print(s2)
中华人民共和国，中华人民共和国
```

7.4.5 replace()、maketrans()、translate()

- 字符串对象的`maketrans()`方法用来生成字符映射表，`translate()`方法用来根据映射表中定义的对应关系转换字符串并替换其中的字符，使用这两个方法的组合可以同时处理多个字符。

```
>>> table = ''.maketrans('0123456789', '〇一二三四五六七八九')
>>> '2021年10月30日'.translate(table)
'二〇二一年一〇月三〇日'
```

7.4.5 replace()、maketrans()、translate()

- 应用：凯撒加密，每个字母替换为后面第k个。

```
>>> import string
>>> def kaisa(s, k):
    lower = string.ascii_lowercase           # 小写字母
    upper = string.ascii_uppercase          # 大写字母
    before = string.ascii_letters
    after = lower[k:] + lower[:k] + upper[k:] + upper[:k]
    table = ''.maketrans(before, after)      # 创建映射表
    return s.translate(table)

>>> s = "Python is a great programming language. I like it!"
>>> kaisa(s, 3)
'Sbwkrq lv d juhdu surjudpplqj odqjxdjh. L olnh lw!'
```

7.4.6 strip()、rstrip()、lstrip()

- strip(chars=None, /)、rstrip(chars=None, /)、lstrip(chars=None, /)
用于删除字符串两侧、右侧、左侧的空白字符或指定的字符，返回处理后的新字符串。

```
>>> s = ' abc '
>>> s.strip()
'abc'
# 删除空白字符

>>> '\n\nhello world \n\n'.strip()
'hello world'
# 删除空白字符

>>> 'aaaassddf'.strip('a')
'ssddf'
# 删除指定的字符

>>> 'aaaassddf'.strip('af')
'ssdd'
# 删除两侧的字符a和f

>>> 'aaaassddf'aaa'.rstrip('a')
'aaaassddf'
# 删除字符串右端指定的字符

>>> 'aaaassddf'aaa'.lstrip('a')
'ssddf'aaa'
# 删除字符串左端指定的字符
```

7.4.6 strip()、rstrip()、lstrip()

- 这三个方法的参数指定的字符串不作为一个整体对待，而是在原字符串的两侧、右侧、左侧删除参数字符串中包含的所有字符，一层一层地从外往里扒。

```
>>> 'aabbccddeeffg'.strip('af') # 字母f不在字符串两侧，所以不删除
'bbccddeeffg'
>>> 'aabbccddeeffg'.strip('gaf')
'bbccdeeee'
>>> 'aabbccddeeffg'.strip('gaef')
'bbccdd'
>>> 'aabbccddeeffg'.strip('gbaef')
'ccdd'
>>> 'aabbccddeeffg'.strip('gbaefcd')
''
```

7.4.7 startswith()、endswith()

■ `s.startswith(t)`、`s.endswith(t)`，判断字符串是否以指定字符串开始或结束

```
>>> s = 'Beautiful is better than ugly.'
>>> s.startswith('Be')           # 检测整个字符串
True
>>> s.startswith('Be', 5)        # 指定检测范围起始位置
False
>>> s.startswith('Be', 0, 5)     # 指定检测范围起始和结束位置
True
>>> import os
>>> [filename
    for filename in os.listdir(r'c:\\')
    if filename.endswith(('.bmp', '.jpg', '.gif'))]
# 列表C盘根目录下所有bmp、jpg、gif文件名
```

7.4.8 isalnum()、isalpha()、isdigit()、isdecimal()、isnumeric()、isspace()、isupper()、islower()

- isalnum()、isalpha()、isdigit()、isdecimal()、isnumeric()、isspace()、isupper()、islower()，用来测试字符串是否全部为数字或字母、是否全部为字母、是否全部为数字字符、是否全部为空白字符、是否全部为大写字母、是否为小写字母。

```
>>> '1234abcd'.isalnum()      # 只包含阿拉伯数字或英文字母时返回True
True
>>> '1234abcd'.isalpha()      # 全部为英文字母时返回True
False
>>> '1234abcd'.isdigit()      # 全部为数字时返回True
False
>>> 'abcd'.isalpha()
True
>>> '1234.0'.isdigit()
False
```

7.4.8 isalnum()、isalpha()、isdigit()、isdecimal()、isnumeric()、isspace()、isupper()、islower()

```
>>> '1234'.isdigit()
True
>>> '九'.isnumeric()                # isnumeric()方法支持汉字数字
True
>>> '九'.isdigit()
False
>>> '九'.isdecimal()
False
>>> 'IVIII X'.isdecimal()
False
>>> 'IVIII X'.isdigit()
False
>>> 'IVIII X'.isnumeric()           # 支持罗马数字
True
```


7.4.9 center()、ljust()、rjust()

- `center(width, fillchar=' ', /)`、`ljust(width, fillchar=' ', /)`、`rjust(width, fillchar=' ', /)`，返回指定宽度的新字符串，原字符串居中、居左或居右出现在新字符串中，如果指定宽度大于字符串长度，则使用指定的字符（默认为空格）进行填充。

```
>>> 'Hello world!'.center(20)          # 居中，两侧以空格进行填充
'      Hello world!      '
>>> 'Hello world!'.center(20, '=')      # 居中，两侧以字符=进行填充
'====Hello world!===='
>>> 'Hello world!'.ljust(20, '=')       # 居左，右侧以字符=进行填充
'Hello world!===== '
>>> 'Hello world!'.rjust(20, '=')       # 居右，左侧以字符=进行填充
'=====Hello world!'
```

7.4.10 字符串对象支持的运算符

- Python字符串支持加法运算符，表示两个字符串连接，生成新字符串。

```
>>> 'hello ' + 'world'  
'hello world'
```

7.4.10 字符串对象支持的运算符

- Python字符串支持与**整数**的乘法运算，表示序列重复，也就是**字符串内容的重复，得到新字符串**。

```
>>> 'abcd' * 3
```

```
'abcdabcdabcd'
```

```
>>> from itertools import repeat
```

```
>>> repeat('Python', 3)
```

```
repeat('Python', 3)
```

```
>>> ''.join(repeat('Python', 3))
```

```
'PythonPythonPython'
```

7.4.10 字符串对象支持的运算符

■ 成员判断，关键字in

```
>>> 'a' in 'abcde'      # 测试一个字符中是否存在于另一个字符串中
```

```
True
```

```
>>> 'ab' in 'abcde'
```

```
True
```

```
>>> 'ac' in 'abcde'      # 关键字in左边的字符串作为一个整体对待
```

```
False
```

```
>>> 'j' in 'abcde'
```

```
False
```

7.4.5 replace()、maketrans()、translate()

- 应用：测试用户输入中是否有敏感词，如果有的话就把敏感词替换为3个星号***。

```
>>> words = ('测试', '非法', '暴力', '话')
>>> text = '这句话里含有非法内容'
>>> for word in words:
    if word in text:
        text = text.replace(word, '***')
>>> text
'这句***里含有***内容'
```

7.4.11 适用于字符串对象的内置函数

```
>>> x = 'Hello world.'
>>> list(zip(x,x))                # 组合对应位置上的字符
[('H', 'H'), ('e', 'e'), ('l', 'l'), ('l', 'l'), ('o', 'o'), (' ', ' '), ('w', 'w'),
 ('o', 'o'), ('r', 'r'), ('l', 'l'), ('d', 'd'), ('.', '.')]
>>> sorted(x)                    # 按字符Unicode编码排序
[' ', '.', 'H', 'd', 'e', 'l', 'l', 'l', 'o', 'o', 'r', 'w']
>>> list(reversed(x))            # 翻转
['.', 'd', 'l', 'r', 'o', 'w', ' ', 'o', 'l', 'l', 'e', 'H']
>>> list(enumerate(x))           # 枚举字符串中的字符
[(0, 'H'), (1, 'e'), (2, 'l'), (3, 'l'), (4, 'o'), (5, ' '), (6, 'w'), (7, 'o'), (8,
 'r'), (9, 'l'), (10, 'd'), (11, '.')]
>>> list(map(lambda i,j: i+j, x, x))
['HH', 'ee', 'll', 'll', 'oo', ' ', ' ', 'ww', 'oo', 'rr', 'll', 'dd', '..']
>>> len(x)                       # 字符串长度
12
>>> max(x)                       # 最大字符
'w'
>>> min(x)                       # 最小字符
'.'
```

7.4.11 适用于字符串对象的内置函数

- 内置函数`eval()`用来把任意字符串转化为Python表达式并进行求值。

```
>>> eval('3+4')
```

```
7
```

计算表达式的值

```
>>> eval('[1, 2, 3, 4]')
```

```
[1, 2, 3, 4]
```

还原字符串中的复杂数据类型

```
>>> eval('{"a": 97, "b": 98}')
```

```
{'a': 97, 'b': 98}
```

```
>>> a = 3
```

```
>>> b = 5
```

```
>>> eval('a+b')
```

```
8
```

要求变量a和b已存在，否则出错

```
>>> import math
```

```
>>> eval('math.sqrt(3)')
```

```
1.7320508075688772
```

7.4.11 适用于字符串对象的内置函数

- Python的内置函数`eval()`可以计算任意合法表达式的值，如果有用户巧妙地构造并输入恶意字符串，可以执行任意外部程序或者实现其他目的，例如下面的代码运行后可以启动记事本程序：

```
>>> a = input('Please input a value:')  
Please input a value: __import__('os').startfile(r'C:\Windows\notepad.exe')  
>>> eval(a)
```

- 下面的代码则会导致屏幕一闪，那一瞬间在当前文件夹中创建了一个子文件夹
`testtest`:

```
>>> eval("__import__('os').system('md testtest')")
```


7.4.11 适用于字符串对象的内置函数

- 对字符串中的内容进行类型转换时，建议使用安全函数`ast.literal_eval()`。

```
>>> import ast
```

```
>>> ast.literal_eval('[1, 2, 3, 4]')
```

```
[1, 2, 3, 4]
```

```
>>> ast.literal_eval('3+5')
```

```
ValueError: malformed node or string: <_ast.BinOp object at 0x000002427F4E3EE0>
```

```
>>> ast.literal_eval(r"__import__('os').startfile(r'C:\Windows\notepad.exe')")
```

```
ValueError: malformed node or string: <_ast.Call object at 0x000002427F4FF9D0>
```

7.4.12 字符串对象的切片操作

- 切片也适用于字符串，但只能用来读取其中的元素，不支持字符串修改。

```
>>> 'Explicit is better than implicit.'[:8]
```

```
'Explicit'
```

```
>>> 'Explicit is better than implicit.'[9:23]
```

```
'is better than'
```

7.5 字符串常量

- Python标准库string中定义数字字符、标点符号、英文字母、大写字母、小写字母等常量。

```
>>> import string
>>> string.digits
'0123456789'
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

7.5 字符串常量

- 应用：生成指定长度的随机密码。 [code\7.5.py](#)

```
from random import choices
from string import ascii_letters, digits

characters = ascii_letters + digits + ',._'

def generate_password(length):
    return ''.join(choices(characters, k=length))

print(generate_password(10))
print(generate_password(10))
```

7.6 中英文分词

```
>>> import jieba                                # 导入jieba模块
>>> x = '分词的准确度直接影响了后续文本处理和挖掘算法的最终效果。'
>>> jieba.cut(x)                                # 使用默认词库进行分词，使用lcut()函数可以直接得到列表
<generator object Tokenizer.cut at 0x00000000342C990>
>>> list(_)
['分词', '的', '准确度', '直接', '影响', '了', '后续', '文本处理', '和', '挖掘', '算法', '的', '最终', '效果', '。']
>>> list(jieba.cut('纸杯'))
['纸杯']
>>> list(jieba.cut('花纸杯'))
['花', '纸杯']
>>> jieba.add_word('花纸杯')                    # 增加词条
>>> list(jieba.cut('花纸杯'))                  # 使用新词库进行分词
['花纸杯']
>>> import snownlp                              # 导入snownlp模块
>>> snownlp.SnowNLP('学而时习之，不亦说乎').words
['学而', '时习', '之', '，', '，', '不亦', '说乎']
```

7.7 汉字到拼音的转换

```
>>> from pypinyin import lazy_pinyin, pinyin
>>> lazy_pinyin('董付国')                # 返回拼音
['dong', 'fu', 'guo']
>>> lazy_pinyin('董付国', 1)              # 带声调的拼音
['dǒng', 'fù', 'guó']
>>> lazy_pinyin('董付国', 2)              # 另一种拼音形式，数字表示前面字母的声调
['do3ng', 'fu4', 'guo2']
>>> lazy_pinyin('董付国', 3)              # 只返回拼音首字母
['d', 'f', 'g']
>>> lazy_pinyin('重要', 1)                # 能够根据词组智能识别多音字
['zhòng', 'yào']
>>> lazy_pinyin('重阳', 1)
['chóng', 'yáng']
>>> pinyin('重阳')                        # 返回拼音
[['chóng'], ['yáng']]
>>> pinyin('重阳节', heteronym=True)      # 返回多音字的所有读音
[['chóng'], ['yáng'], ['jié', 'jiē']]
>>> x = '山东烟台的大樱桃真好吃啊'
>>> sorted(x, key=lambda ch: lazy_pinyin(ch)) # 按拼音对汉字进行排序
['啊', '吃', '大', '的', '东', '好', '山', '台', '桃', '烟', '樱', '真']
```

7.8 精彩案例赏析

- **例7-1** 编写函数实现字符串加密和解密，循环使用指定密钥，采用异或算法。

[code\例7-1.py](#)

```
from itertools import cycle
```

```
def convert(source, key):
```

```
    # 对应位置上字符的Unicode编码进行异或运算，异或运算两次可得到原立，即A^B^B=A
```

```
    return ''.join(map(lambda s, k: chr(ord(s) ^ ord(k)), source, cycle(key)))
```

```
source = '微信公众号：Python小屋。'
```

```
key = '董付国'
```

```
encrypted = convert(source, key)
```

```
decrypted = convert(encrypted, key)
```

```
print(f'原文：{source}\n加密后：{encrypted}\n解密后：{decrypted}')
```

原文：微信公众号：Python小屋。

加密后：ϯŁεϰϯⓂ萑亡嚟萋廉嘹ϯϯ替

解密后：微信公众号：Python小屋。

7.8 精彩案例赏析

- **例7-2** 编写程序，生成大量随机信息，这在需要获取大量数据来测试或演示软件功能的时候非常有用，不仅能真实展示软件功能或算法，还可以避免泄露真实数据或者引起不必要的争议。

[code\例7-2.py](#)

7.8 精彩案例赏析

■ **例7-3** 检查并判断密码字符串的安全强度。 [code\例7-3.py](#)

```
import string

def check(pwd):
    # 密码必须至少包含6个字符
    if not isinstance(pwd, str) or len(pwd)<6:
        return 'not suitable for password'
    # 密码强度等级与包含字符种类的对应关系
    d = {1:'weak', 2:'below middle', 3:'above middle', 4:'strong'}
    # 分别用来标记pwd是否含有数字、小写字母、大写字母和指定的标点符号
    r = [False] * 4
```

7.8 精彩案例赏析

```
for ch in pwd:
    # 是否包含数字
    if not r[0] and ch in string.digits:
        r[0] = True
    # 是否包含小写字母
    elif not r[1] and ch in string.ascii_lowercase:
        r[1] = True
    # 是否包含大写字母
    elif not r[2] and ch in string.ascii_uppercase:
        r[2] = True
    # 是否包含指定的标点符号
    elif not r[3] and ch in ',.!?;<>':
        r[3] = True
# 统计包含的字符种类，返回密码强度
return d.get(r.count(True), 'error')
```

```
print(check('a2Cd,'))
```