Chapter 1

1.1 用正则表达式查找文本模式

向 re.compile()传入一个字符串值，表示正则表达式，它将返回一个 Regex 模式对象（或者就简称为 Regex 对象）。

Regex 对象的 search()方法查找传入的字符串，寻找该正则表达式的所有匹配。如果字符串中没有找到该正则表达式模式，search()方法将返回 None。如果找到了该模式，search()方法将返回一个 Match 对象。Match 对象有一个 group()方法，它返回被查找字符串中实际匹配的文本。

虽然在 Python 中使用正则表达式有几个步骤，但每一步都相当简单。

1. 用 import re 导入正则表达式模块。
2. 用 re.compile()函数创建一个 Regex 对象（记得使用原始字符串）。
3. 向 Regex 对象的 search()方法传入想查找的字符串。它返回一个 Match 对象。
4. 调用 Match 对象的 group()方法，返回实际匹配文本的字符串。

1.2 用正则表达式匹配更多模式

1.2.1 利用括号分组

> 我觉得可以不翻译为分组，而是理解为组件。

添加括号将在正则表达式中创建“分组”：然后可以使用 group()匹配对象方法，从一个分组中获取匹配的文本。正则表达式字符串中的第一对括号是第 1 组。第二对括号是第 2 组。向 group()匹配对象方法传入整数 1 或 2，就可以取得匹配文本的不同部分。向 group()方法传入 0 或不传入参数，将返回整个匹配的文本。**只找第一个不会多找**

如果想要一次就获取所有的分组，请使用 groups()方法。

括号在正则表达式中有特殊的含义，但是如果你需要在文本中匹配括号，怎么办？在这种情况下，就需要用斜杠对 (与) 进行字符转义。

在正则表达式中，以下字符具有特殊含义：

```
. ^ $ * + ? { } ( ) [ ] \ |
```

如果要检测包含这些字符的文本模式，那么就需要用斜杠对它们进行转义。

要确保正则表达式中，没有将转义的括号 (与) 误作为 (与)。如果你收到类似 missing 或 unbalanced parenthesis 的错误信息，则可能是忘记了为分组添加转义的右括号。

1.2.2 用管道匹配多个分组

字符 | 称为“管道”。希望匹配许多表达式中的一个时，就可以使用它。例如，正则表达式 r‘Batman|Tina Fey’ 将匹配 ‘Batman’ 或 ‘Tina Fey’。

如果 Batman 和 Tina Fey 都出现在被查找的字符串中，第一次出现的匹配文本，将作为 Match 对象返回。

> 利用 findall()方法，可以找到“所有”匹配的地方。

1.2.3 用问号实现可选匹配

有时候，想匹配的模式是可选的。就是说，不论这段文本在不在，正则表达式都会认为匹配。字符 ? 表明它前面的分组在这个模式中是可选的。你可以认为 ? 是在说，“匹配这个问号之前的分组零次或一次”。

1.2.4 用星号匹配零次或多次

*（称为星号）意味着“匹配零次或多次”，即星号之前的分组，可以在文本中出现任意次。它可以完全不存在，或一次又一次地重复。

1.2.5 用加号匹配一次或多次

* 意味着“匹配零次或多次”，+（加号）则意味着“匹配一次或多次”。星号不要求分组出现在匹配的字符串中，但加号不同，加号前面的分组必须“至少出现一次”。这不是可选的。

1.2.6 用花括号匹配特定次数

如果想要一个分组重复特定次数，就在正则表达式中该分组的后面，跟上花括号包围的数字。除了一个数字，还可以指定一个范围，即在花括号中写下一个最小值、一个逗号和一个最大值。也可以不写花括号中的第一个或第二个数字，不限定最小值或最大值。

1.3 贪心和非贪心匹配

Python 的正则表达式默认是“贪心”的，这表示在有二义的情况下，它们会尽可能匹配最长的字符串。花括号的“非贪心”版本匹配尽可能最短的字符串，即在结束的花括号后跟着一个问号。

> 请注意，问号在正则表达式中可能有两种含义：声明非贪心匹配或表示可选的分组。这两种含义是完全无关的。

1.4 findall()方法

除了 search 方法外，Regex 对象也有一个 findall() 方法。search() 将返回一个Match 对象，包含被查找字符串中的“第一次”匹配的文本，而 findall() 方法将返回一组字符串，包含被查找字符串中的所有匹配。

Table 1.1: 常用字符分类的缩写代码

缩写字符分类	表示
\d	0 到 9 的任何数字
\D	除 0 到 9 的数字以外的任何字符
\w	任何字母、数字或下划线字符（可以认为是匹配“单词”字符）
\W	除字母、数字和下划线以外的任何字符
\s	空格、制表符或换行符（可以认为是匹配“空白”字符）
\S	除空格、制表符和换行符以外的任何字符

另一方面，findall()不是返回一个 Match 对象，而是返回一个字符串列表，前提是正则表达式没有分组（组件）。列表中的每个字符串都是一段被查找的文本，它匹配该正则表达式。

1.5 字符分类

有许多这样的“缩写字符分类”，如 Table 1.1 所示。

1.6 建立自己的字符分类

你可以用方括号定义自己的字符分类。也可以使用短横表示字母或数字的范围。例如，字符分类 [a-zA-Z0-9] 将匹配所有小写字母、大写字母和数字。请注意，在方括号内，普通的正则表达式符号不会被解释。这意味着，你不需要前面加上反斜杠转义。

注意：这是逻辑上或的关系

通过在字符分类的左方括号后加上一个插入字符（^）就可以得到“非字符类”（逻辑非）。

1.7 插入字符和美元字符

可以在正则表达式的开始处使用插入符号（^），表明匹配必须发生在被查找文本开始处。类似地，可以再正则表达式的末尾加上美元符号（$），表示该字符串必须以这个正则表达式的模式结束。可以同时使用 ^ 和 $，表明整个字符串必须匹配该模式，也就是说，只匹配该字符串的某个子集是不够的。

1.8 通配字符

在正则表达式中，.（句点）字符称为“通配符”。它匹配除了换行之外的所有字符。要记住，句点字符只匹配一个字符。

1.8.1 用点-星匹配所有字符

句点字符表示“除换行外所有单个字符”，星号字符表示“前面字符出现零次或多次”。

点-星使用“贪心”模式：它总是匹配尽可能多的文本。要用“非贪心”模式匹配所有文本，就使用点-星和问号。像和大括号一起使用时那样，问号告诉 Python 用非贪心模式匹配。

1.8.2 用句点字符匹配换行

点-星将匹配除换行外的所有字符。通过传入 re.DOTALL 作为 re.compile() 的第二个参数，可以让句点字符匹配所有字符，包括换行字符。

1.9 正则表达式符号复习

- ?匹配零次或一次前面的分组。
- *匹配零次或多次前面的分组。
- +匹配一次或多次前面的分组。
- {n}匹配 n 次前面的分组。
- {n,}匹配 n 次或更多前面的分组。
- {,m}匹配零次到 m 次前面的分组。
- {n,m}匹配至少 n 次、至多 m 次前面的分组。
- {n,m}? 或 *? 或 +? 对前面的分组进行非贪心匹配。
- ^spam 意味着字符串必须以 spam 开始。
- spam$ 意味着字符串必须以 spam 结束。
- . 匹配所有字符，换行符除外。
- [abc] 匹配方括号内的任意字符（诸如 a、b 或 c）。
- [^abc] 匹配不在方括号内的任意字符。
- \d、\w 和 \s 分别匹配数字、单词和空格。
- \D、\W 和 \S 分别匹配出数字、单词和空格外的所有字符。

1.10 不区分大小写的匹配

有时候你只关心匹配字母，不关心它们是大写或小写。要让正则表达式不区分大小写，可以向 re.compile()传入 re.IGNORECASE 或 re.I，作为第二个参数。

1.11 用 sub()方法替换字符串

正则表达式不仅能找到文本模式，而且能够用新的文本替换掉这些模式。Regex 对象的 sub()方法需要传入两个参数。第一个参数是一个字符串，用于取代发现的匹配。第二个参数是一个字符串，即正则表达式。sub()方法返回替换完成后的字符串。

有时候，你可能需要使用匹配的文本本身，作为替换的一部分。在 sub()的第一个参数中，可以输入\1、\2、\3...。表示“在替换中输入分组 1、2、3...的文本”。

1.12 管理复杂的正则表达式

如果要匹配的文本模式很简单，正则表达式就很好。但匹配复杂的文本模式，可能需要长的、费解的正则表达式。你可以告诉 re.compile()，忽略正则表达式字符串中的空白符和注释，从而缓解这一点。要实现这种详细模式，可以向 re.compile()传入变量 re.VERBOSE，作为第二个参数。

你可以将正则表达式放在多行中，并加上注释正则表达式字符串中的注释规则，与普通的 Python 代码一样：# 符号和它后面直到行末的内容，都被忽略。而且，表示正则表达式的多行字符串中，多余的空白字符也不认为是要匹配的文本模式的一部分。这让你能够组织正则表达式，让它更可读。

1.13 组合使用 re.IGNOREC ASE、re.DOTALL 和 re.VERBOSE

如果你希望在正则表达式中使用 re.VERBOSE 来编写注释，还希望使用 re.IGNORECASE 来忽略大小写，该怎么办？遗憾的是，re.compile()函数只接受一个值作为它的第二参数。可以使用管道

字符（|）将变量组合起来，从而绕过这个限制。管道字符在这里称为“按位或”操作符。