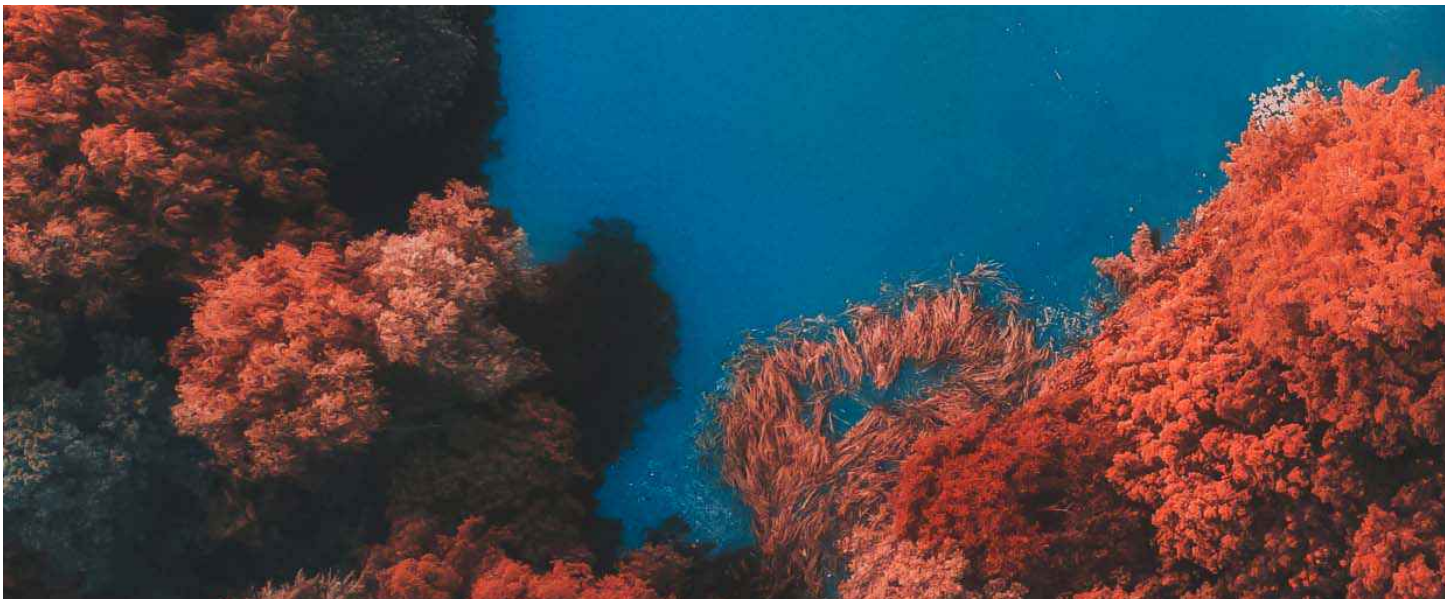


正则表达式

> https://regexr-cn.com/
>
> https://codejiaonang.com/#/

[]

- 匹配

-

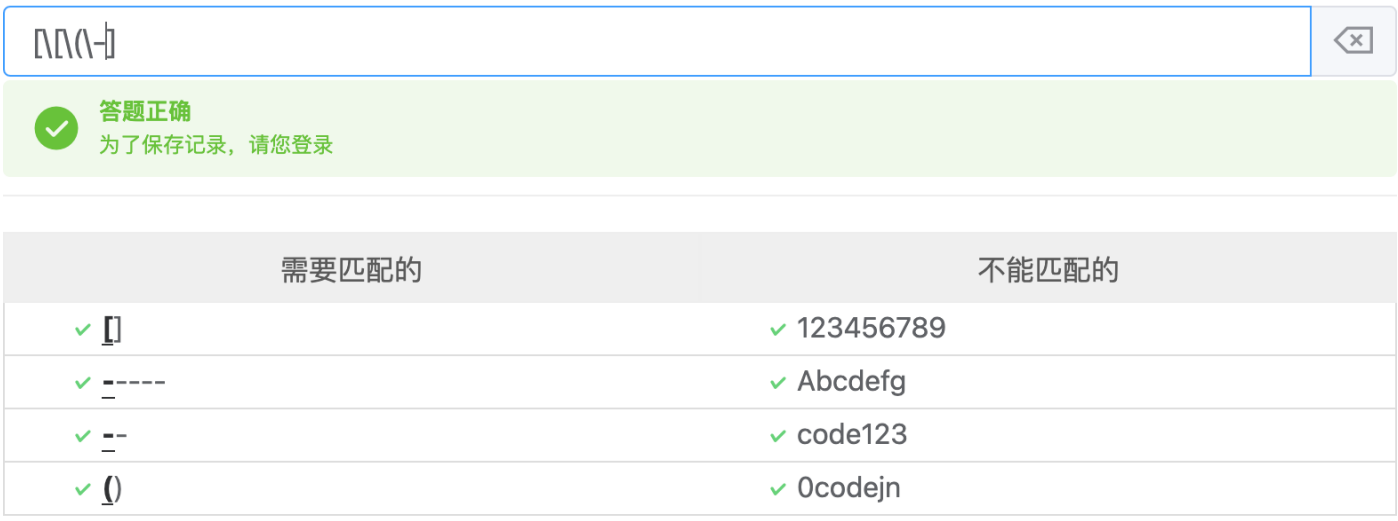
- 区间

a. 要匹配任意数字可以使用 `[0-9]`；
b. 如果想要匹配所有小写字母，可以写成 `[a-z]`；
c. 想要匹配所有大写字母可以写成 `[A-Z]`

这个时候我们需要对 `-` 号进行转义操作，即 `\-`。

在正则中使用 `\` 就可以进行对特殊符号进行转义，对 `-` 进行转义就可以表示为 `\-`，即 `\-` 就代表了 `-` 号本身。

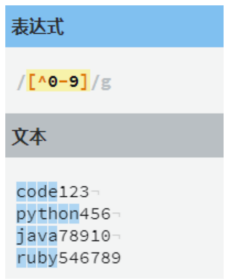
> 转义符 `\` 也适用于其他的符号，例如匹配圆括号可以使用 `\(`



^

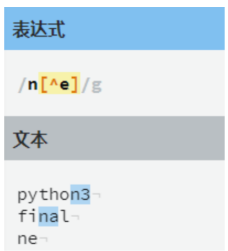
取反

例如：匹配不包含数字的字符组



可以通过在字符数组开头使用 `^` 字符实现取反操作，从而可以反转一个字符组（意味着会匹配任何指定字符之外的所有字符）。

再看一个例子：



这里的 `n[^e]` 的意思就是 `n` 后面的字母不能为 `e`。

快捷方式

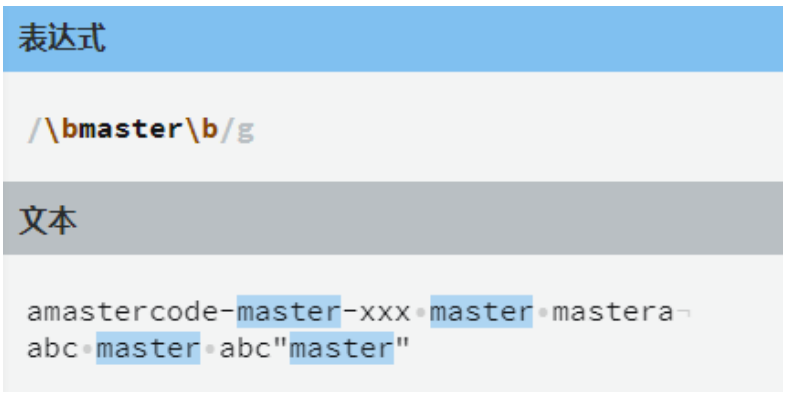
正则表达式引擎提供了一些快捷方式如： `\w` 可以与任意单词字符匹配。

当我们想要匹配任意数字的时候也可以使用快捷方式 `\d`， `d` 即 `digit` 数字的意思，等价于 `[0-9]`。

	A	B
1	**快捷方式**	**描述**
2	\w	与任意单词字符匹配，任意单词字符表示 [A-Z]、 [a-z]、[0-9]、_
3	\d	与任意数字匹配

`\s` 快捷方式可以匹配空白字符，比如空格，tab、换行等。

`\b` 匹配的是单词的边界，例如，



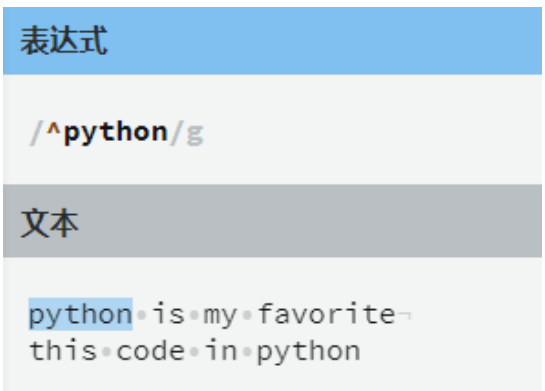
`\bmaster\b` 就仅匹配有边界的 `master` 单词。

快捷方式也可以取反，例如对于 `\w` 的取反为 `\W` ，将小写改写成大写即可，其他快捷方式也遵循这个规则。

开始和结束

正则表达式中 `^` 指定的是一个字符串的开始， `$` 指定的是一个字符串的结束。

例如：



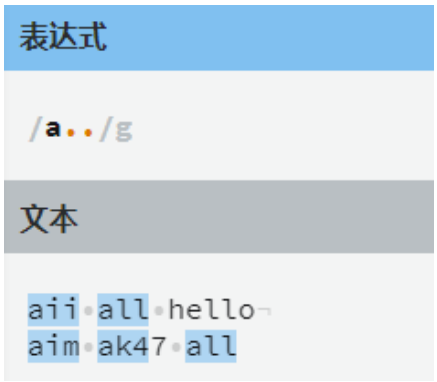
指定字符串的结束：

. 任意字符

`.` 字符代表匹配任何单个字符，它只能出现在方括号以外。

值得注意的是： `.` 字符只有一个不能匹配的字符，也就是换行符（`\n`），不过要让 `.` 字符与换行符匹配也是可以的，以后会讨论。

示例：

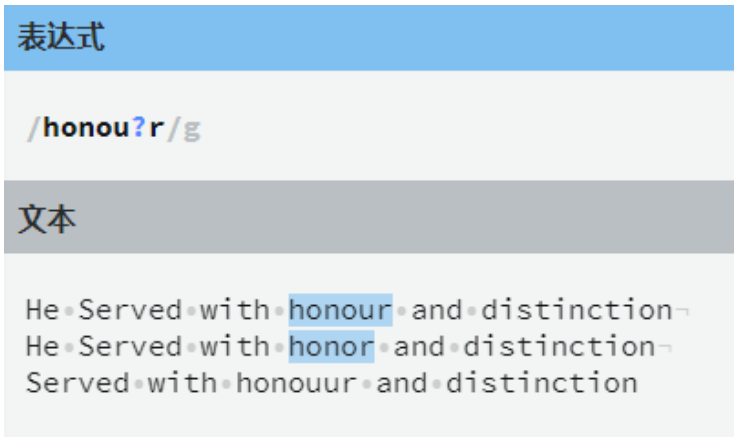


？ 可选字符

有时，我们可能想要匹配一个单词的不同写法，比如 `color` 和 `colour` ，或者 `honor` 与 `honour` 。

这个时候我们可以使用 `?` 符号指定一个字符、字符组或其他基本单元可选，这意味着正则表达式引擎将会期望该字符出现零次或一次。

例如：



在这里 `u?` 表示 `u` 是可选的，即可以出现也可以不出现，可以匹配的是 `honor` 和 `honour` 。

通过这个案例可以知道 `?` 的作用就是匹配它之前的字符 `0` 次或 `1` 次。

> 请你思考一个问题： `.?` 表达式能匹配什么呢?
>
> 任意一个字符 0次 或 1次

简单说是贪婪匹配与非贪婪匹配的区别。

比如说匹配输入串A: 101000000000100

使用 1.*1 将会匹配到1010000000001, 匹配方法: 先匹配至输入串A的最后, 然后向前匹配, 直到可以匹配到1, 称之为贪婪匹配。

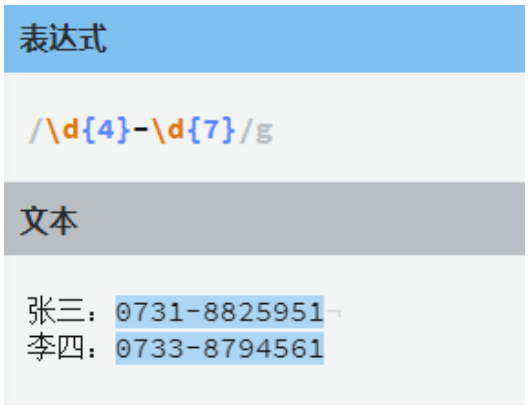
使用 1.*?1 将会匹配到101, 匹配方法: 匹配下一个1之前的所有字符, 称之为非贪婪匹配。

所有带有量词的都是非贪婪匹配: .*?, .+?, .{2,6}? 甚至 .??

重复

在一个字符组后加上 `{N}` 就可以表示在它之前的字符组出现 `N` 次。

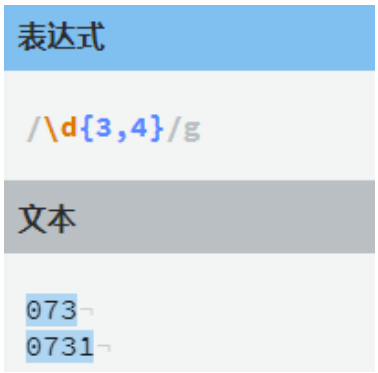
例如：



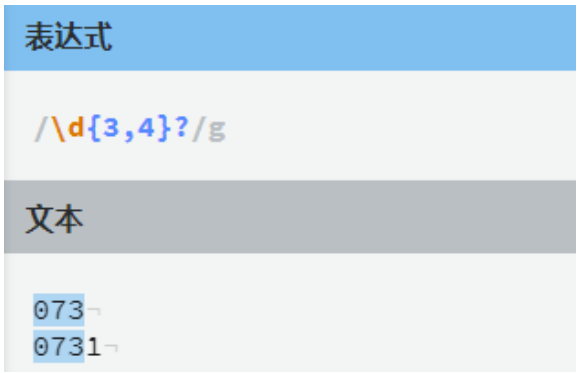
可能有时候，我们不知道具体要匹配字符组要重复的次数，比如身份证有 `15` 位也有 `18` 位的。

这里重复区间就可以出场了，语法：`{M,N}`，`M` 是下界而 `N` 是上界。

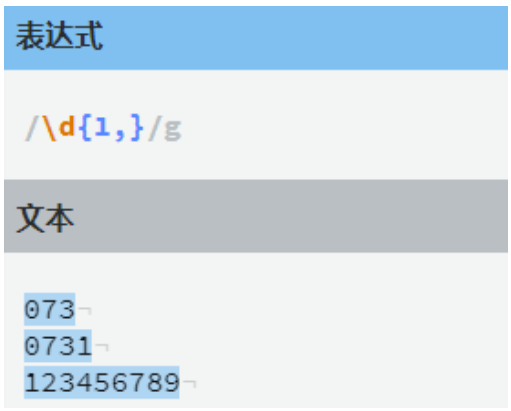
举个例子：



`\d{3,4}` 既可以匹配 `3` 个数字也可以匹配 `4` 个数字，不过当有 `4` 个数字的时候，优先匹配的是 `4` 个数字，这是因为正则表达式默认是贪婪模式，即尽可能的匹配更多字符，而要使用非贪婪模式，我们要在表达式后面加上 `?` 号。



有时候我们可能遇到字符组的重复次数没有边界，例如：



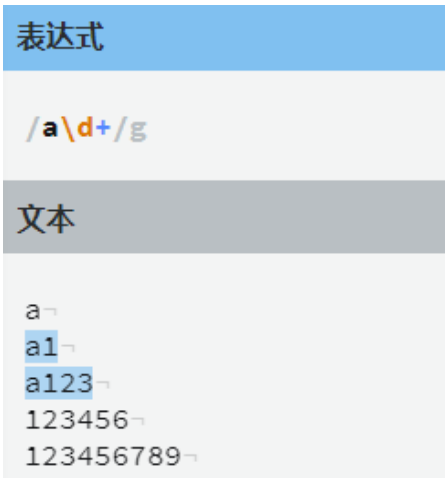
闭区间不写即可表示匹配一个或无数个。

速写

还可以使用两个速写字符指定常见的重复情况，可以使用 `+` 匹配 `1` 个到无数个，使用 `*` 代表 `0` 个到无数个。

即： `+` 等价于 `{1,}` ， `*` 等价于 `{0,}` 。

`+` 号示例：



`*` 号示例：

请使用正则表达式匹配以 `f` 开头的数据。

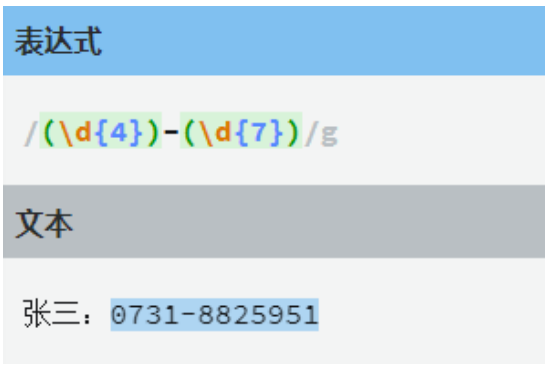
f.+

分组

在正则表达式中还提供了一种将表达式分组的机制，当使用分组时，除了获得整个匹配。还能够在匹配中选择每一个分组。

要实现分组很简单，使用 `()` 即可。

例如：



这段正则表达式将文本分成了两组，第一组为： `0731` ，第二组为 `8825951` 。

Group 1	n/a	0731
Group 2	n/a	8825951

分组有一个非常重要的功能—— `捕获数据` 。所以 `()` 被称为捕获分组，用来捕获数据，当我们想要从匹配好的数据中提取关键数据的时候可以使用分组。

`(\d{4})` 和 `(\d{7})` 就分别捕获了两段数据：

1. `0731`
2. `8825951`

使用分组的同时还可以使用 或者（ `or` ）条件。

例如要提取所有图片文件的后缀名，可以在各个后缀名之间加上一个 `|` 符号：

表达式

```
/(.jpg|.gif|.jpeg|.png)/g
```

文本

```
image.jpg
image.jpeg
image.png
image.gif
not_image.txt
not_image.doc
not_image.xls
not_image.ppt
```

有时候，我们并不需要捕获某个分组的内容，但是又想使用分组的特性。

这个时候就可以使用非捕获组`(?:表达式)`，从而不捕获数据，还能使用分组的功能。

例如想要匹配两个字母组成的单词或者四个字母组成的单词就可以使用非捕获分组：

表达式

```
/\b(?:\w{2}|\w{4})\b/g
```

文本

```
he
ha
code
hello
master
capsule
```

获取电话号码

提取目标数据中的电话号码

(?:.*)[\-:](\d{5})

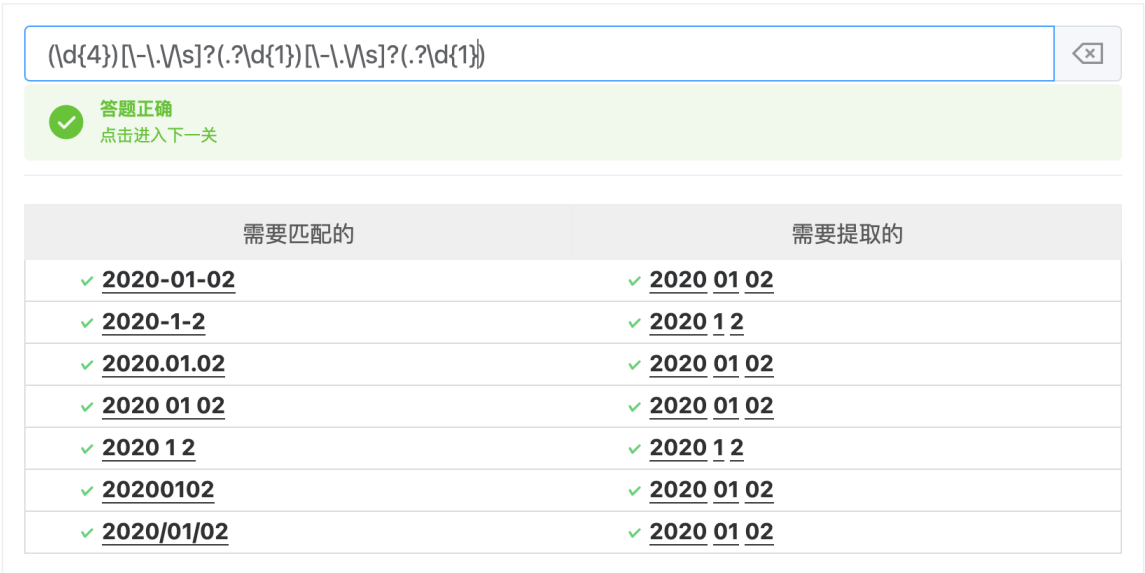
答题正确
点击进入下一关

需要匹配的	需要提取的
✓ 01-75855	✓ 75855
✓ 0731-75855	✓ 75855
✓ 12345-75855	✓ 75855
✓ tel:75855	✓ 75855

提取年月日的数据

接下来请提取所有年月日的数据

> 注意：因为正则表达式中 `()` 代表分组，所以如果要匹配 `()` 就需要将使用转义符 `\`，例如要匹配 `)` 就要写成 `\)`。

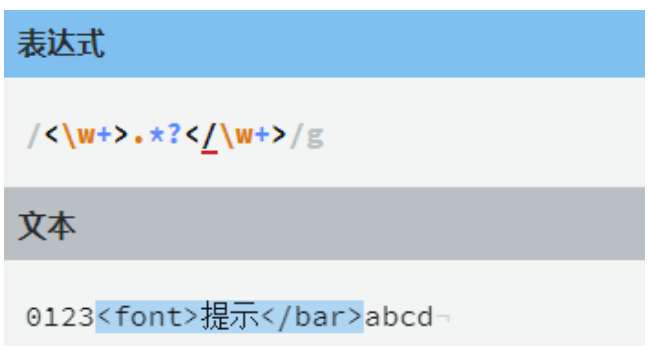


需要匹配的	需要提取的
2020-01-02	2020 01 02
2020-1-2	2020 1 2
2020.01.02	2020 01 02
2020 01 02	2020 01 02
2020 1 2	2020 1 2
20200102	2020 01 02
2020/01/02	2020 01 02

正则表达式还提供了一种引用之前匹配分组的机制，有些时候，我们或许会寻找到一个子匹配，该匹配接下来会再次出现。

例如，要匹配一段 HTML 代码，比如：`0123<font>提示</font>abcd`，可能会编写出这样一段正则表达式：

这确实可以匹配，不过可能还有另一种情况，如果数据改成这样：`<font>提示</bar>`



在这里 `font` 和 `bar` 明显不是一对正确的标签，但是我们编写的正则表达式还是将它们给匹配了，所以这个结果是错误的。

我们想让后面分组的正则也匹配 `font` ，但是现在所有形式的都会匹配。

那如果想让后面分组的正则和第一个分组的正则匹配同样的数据该如何做呢?

可以使用分组的回溯引用，使用 `\N` 可以引用编号为 `N` 的分组，因此上述例子的代码我们可以改为：

通过这个例子，可以发现 `\1` 表示的就是第一个分组，在这里第一个分组匹配的是 `font` 所以 `\1` 就代表 `font` 。

匹配单词

接下来请你编写代码匹配符合 `ab` `ba` 这种关系的单词

(\w{1})(\w{1})\2\1

答题正确
点击进入下一关

需要匹配的	不能匹配的
✓ a**bccb**a	✓ aaruria
✓ **alla**gmatic	✓ bluted
✓ **otto**	✓ cffusive
✓ **abba**	✓ dusoid
✓ a**sffs**	✓ eesthophysiology
✓ **maam**	✓ fmphimictical
✓ w**arra**ndice	✓ ccritan

请编写正则表达式匹配下列有重复的数据：

(\w.+)\1

答题正确
点击进入下一关

需要匹配的	不能匹配的
mama	abcd
baba	abba
froufrou	gummage
barbar	asdhweuir
haha	babb
hehe	aa
ohhohh	ccbbdd

先行断言

很多人也称先行断言和后行断言为环视，也有人叫预搜索，其实叫什么无所谓，重要的是知道如何使用它们！

先行断言和后行断言总共有四种：

1. 正向先行断言
2. 反向先行断言
3. 正向后行断言
4. 反向后行断言

正向先行断言：`(?=表达式)`，指在某个位置向右看，表示所在位置右侧必须能匹配`表达式`

例如：

`我喜欢你 我喜欢 我喜欢我 喜欢 喜欢你`

如果要取出喜欢两个字，要求这个喜欢后面有你，这个时候就要这么写：`喜欢(?=你)`，这就是正向先行断言。

表达式

/喜欢(?=你)/g

文本

我喜欢你 我喜欢 我喜欢我 喜欢 喜欢你

提取包含大小写字母的字符串

先行断言可以用来判断字符串是否符合特定的规则，例如提取包含至少一个大小写字母的字符串：

`(?=.*?[a-z])(?=.*?[A-Z]).+` 这段正则表达式规定了匹配的字符串中必须包含至少一个大写和小写的字母。

密码强度验证

现在请你编写正则表达式进行密码强度的验证，规则如下：

- 至少一个大写字母
- 至少一个小写字母
- 至少一个数字
- 至少 8 个字符

左边为需要你的正则需要匹配的，右边的字符串是你的正则不需要匹配的。

```
(?=.*?[a-z])(?=.*?[A-Z])(?=.*?[0-9]).{8}
```

答题正确
点击进入下一关

需要匹配的	不能匹配的
Admin123456	qwe
pZUJLUpTL2	8848
Tnut2eWPN1	123456
wJxpVhVYi3	asd123
UySRo49ps	Adm123
lg7AHzZ0J	Asd123
oYHMDdHCK9	wjleif932
yiyWKQnWo2	admin123
gTZEEkVrj1	123admin
8lj12340as	123asd123
wdfqe#wefDdf444	ADMIN123()

反向先行断言 `(?!表达式)` 的作用是保证右边不能出现某字符。

例如： `我喜欢你 我喜欢 我喜欢我 喜欢 喜欢你`

如果要取出喜欢两个字，要求这个喜欢后面没有你，这个时候就要这么写： `喜欢(?!你)` ，这就是反向先行断言。

匹配标签

编写正则表达式匹配除 `<p>` 或 `</p>` 之外的所有标签。

<(?!p)(\w+)>.*</\1>

答题正确
点击进入下一关

需要匹配的	不能匹配的
** **	
** **	code
** **	
** **	
** **	
** **	
<tr></tr>	
<td></td>	

后行断言

先行断言和后行断言只有一个区别，即先行断言从左往右看，后行断言从右往左看。

正向后行断言：`(?<=表达式)`，指在某个位置向左看，表示所在位置左侧必须能匹配 `表达式`

(?<!\$)\$[^\$]+\$(?!\$)

答题正确
点击进入下一关

需要匹配的	不能匹配的
$ a = a^2 $	abc$asddadd$$
123	$$$ a=b^2 $$
$ A = a / b + b $	abc$asddadd$$
$ a = a^2 $	abc$ a=b^2 $$abc
123	abc$$ a = a**2 $$
$ A = a / b + b $	$$ a=b^2 $$
$ a^{2}+ 3\frac{2}{3} $	abc$asddadd$$
$ aN&123% $**	$$$ a=b^2 $$$$

Exam

实践：匹配所有的小数

请编写正则表达式匹配所有的小数：

^\d+\.\d+$

答题正确
看看其他课程吧

需要匹配的	不能匹配的
0.1	1
3.5	12
1.34	123
2.56	999
1.24	123456
0.123	12.12.12.
77.87	.12
192.168	.10
3.1415926	1.1.1.1