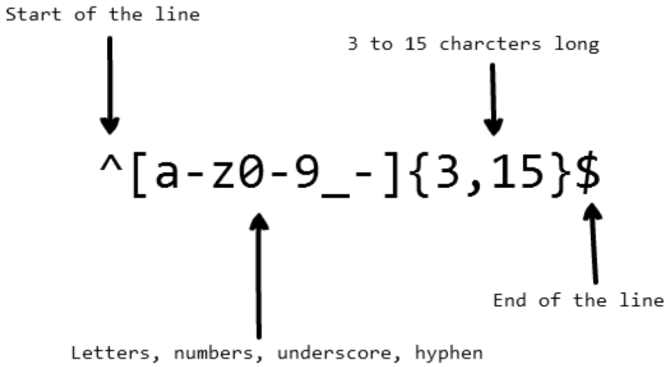
# LEARN REGEX THE EASY WAY

## 什么是正则表达式？

> 正则表达式是一种被用于从文本中检索符合某些特定模式的文本。

正则表达式是从左到右来匹配一个字符串的。"Regular Expression" 这个词太长了，我们通常使用它的缩写 "regex" 或者 "regexp"。正则表达式可以被用来替换字符串中的文本、验证表单、基于模式匹配从一个字符串中提取字符串等等。

想象一下，您正在编写应用程序，并且您希望在用户选择用户名时设置规则。我们希望用户名可以包含字母，数字，下划线和连字符。为了让它看起来不丑，我们还想限制用户名中的字符数量。我们可以使用以下正则表达式来验证用户名:




上面这个正则表达式可以匹配 `john_doe` ， `jo-hn\_doe` 和 `john12\_as` 。但是它不能匹配 `Jo` ，因为该字符串里面包含了大写字符，并且它太短了。

## 目录

## 1. 基本匹配

正则表达式只是我们用于在文本中检索字母和数字的模式。例如正则表达式 `cat`，表示: 字母 `c` 后面跟着一个字母 `a`，再后面跟着一个字母 `t`。

```
"cat" => The cat sat on the mat
```

正则表达式 `123` 会匹配字符串 "123"。通过将正则表达式中的每个字符逐个与要匹配的字符串中的每个字符进行比较，来完成正则匹配。正则表达式通常区分大小写，因此正则表达式 `Cat` 与字符串 "cat" 不匹配。

```
"Cat" => The cat sat on the Cat
```

## 2. 元字符

元字符是正则表达式的基本组成元素。元字符在这里跟它通常表达的意思不一样，而是以某种特殊的含义去解释。有些元字符写在方括号内的时候有特殊含义。元字符如下:

| 元字符 | 描述 |
| --- | --- |
| . | 匹配除换行符以外的任意字符。 |
| [ ] | 字符类，匹配方括号中包含的任意字符。 |
| [^ ] | 否定字符类。匹配方括号中不包含的任意字符 |
| * | 匹配前面的子表达式零次或多次 |
| + | 匹配前面的子表达式一次或多次 |
| ? | 匹配前面的子表达式零次或一次，或指明一个非贪婪限定符。 |
| {n,m} | 花括号，匹配前面字符至少 n 次，但是不超过 m 次。 |
| (xyz) | 字符组，按照确切的顺序匹配字符xyz。 |
| \| | 分支结构，匹配符号之前的字符或后面的字符。 |
| \ | 转义符，它可以还原元字符原来的含义，允许你匹配保留字符 `[ ] ( ) { } . * + ? ^ $ \ \|` |
| ^ | 匹配行的开始 |
| $ | 匹配行的结束 |

## 2.1 英文句号

英文句号 `.` 是元字符的最简单的例子。元字符 `.` 可以匹配任意单个字符。它不会匹配换行符和新行的字符。例如正则表达式 `.ar`，表示: 任意字符后面跟着一个字母 `a`，再后面跟着一个字母 `r`。

```
".ar" => The car parked in the garage.
```

## 2.2 字符集

字符集也称为字符类。方括号被用于指定字符集。使用字符集内的连字符来指定字符范围。方括号内的字符范围的顺序并不重要。例如正则表达式 `[Tt]he`，表示: 大写 `T` 或小写 `t`，后跟字母 `h`，再后跟字母 `e`。

```
"[Tt]he" => The car parked in the garage.
```

然而，字符集中的英文句号表示它字面的含义。正则表达式 `ar[.]`，表示小写字母 `a`，后面跟着一个字母 `r`，再后面跟着一个英文句号 `.` 字符。

```
"ar[.]" => A garage is a good place to park a car.
```

### 2.2.1 否定字符集

一般来说插入字符 `^` 表示一个字符串的开始，但是当它在方括号内出现时，它会取消字符集。例如正则表达式 `[^c]ar`，表示: 除了字母 `c` 以外的任意字符，后面跟着字符 `a`，再后面跟着一个字母 `r`。

```
"[^c]ar" => The car parked in the garage.
```

## 2.3 重复

以下元字符 `+`， `*` 或 `?` 用于指定子模式可以出现多少次。这些元字符在不同情况下的作用不同。

### 2.3.1 星号

该符号 `*` 表示匹配上一个匹配规则的零次或多次。正则表达式 `a*` 表示小写字母 `a` 可以重复零次或者多次。但是它如果出现在字符集或者字符类之后，它表示整个字符集的重复。例如正则表达式 `[a-z]*`，表示: 一行中可以包含任意数量的小写字母。

```
"[a-z]*" => The car parked in the garage #21.
```

该 `*` 符号可以与元符号 `.` 用在一起，用来匹配任意字符串 `.*`。该 `*` 符号可以与空格符 `\s` 一起使用，用来匹配一串空格字符。例如正则表达式 `\s*cat\s*`，表示: 零个或多个空格，后面跟小写字母 `c`，再后面跟小写字母 `a`，再再后面跟小写字母 `t`，后面再跟零个或多个空格。

```
"\s*cat\s*" => The fat cat sat on the cat.
```

### 2.3.2 加号

该符号 `+` 匹配上一个字符的一次或多次。例如正则表达式 `c.+t`，表示: 一个小写字母 `c`，后跟任意数量的字符，后跟小写字母 `t`。

```
"c.+t" => The fat cat sat on the mat.
```

### 2.3.3 问号

在正则表达式中，元字符 `?` 用来表示前一个字符是可选的。该符号匹配前一个字符的零次或一次。例如正则表达式 `[T]?he`，表示: 可选的大写字母 `T`，后面跟小写字母 `h`，后跟小写字母 `e`。

```
"[T]he" => The car is parked in the garage.
```

```
"[T]?he" => The car is parked in the garage.
```

## 2.4 花括号

在正则表达式中花括号(也被称为量词 ?)用于指定字符或一组字符可以重复的次数。例如正则表达式 `[0-9]{2,3}`，表示: 匹配至少2位数字但不超过3位(0到9范围内的字符)。

```
"[0-9]{2,3}" => The number was 9.9997 but we rounded it off to 10.0.
```

我们可以省略第二个数字。例如正则表达式 `[0-9]{2,}`，表示: 匹配2个或更多个数字。如果我们也删除逗号，则正则表达式 `[0-9]{2}`，表示: 匹配正好为2位数的数字。

```
"[0-9]{2,}" => The number was 9.9997 but we rounded it off to 10.0.

"[0-9]{2}" => The number was 9.9997 but we rounded it off to 10.0.
```

## 2.5 字符组

字符组是一组写在圆括号内的子模式 `(...)`。正如我们在正则表达式中讨论的那样，如果我们把一个量词放在一个字符之后，它会重复前一个字符。但是，如果我们把量词放在一个字符组之后，它会重复整个字符组。例如正则表达式 `(ab)*` 表示匹配零个或多个的字符串 "ab"。我们还可以在字符组中使用元字符 `|`。例如正则表达式 `(c|g|p)ar`，表示: 小写字母 `c`、`g` 或 `p` 后面跟字母 `a`，后跟字母 `r`。

```
"(c|g|p)ar" => The car is parked in the garage.
```

## 2.6 分支结构

在正则表达式中垂直条 `|` 用来定义分支结构，分支结构就像多个表达式之间的条件。现在你可能认为这个字符集和分支机构的工作方式一样。但是字符集和分支结构巨大的区别是字符集只在字符级别上有作用，然而分支结构在表达式级别上依然可以使用。例如正则表达式 `(T|t)he|car`，表示: 大写字母 `T` 或小写字母 `t`，后面跟小写字母 `h`，后跟小写字母 `e` 或小写字母 `c`，后跟小写字母 `a`，后跟小写字母 `r`。

```
"(T|t)he|car" => The car is parked in the garage.
```

## 2.7 转义特殊字符

正则表达式中使用反斜杠 `\` 来转义下一个字符。这将允许你使用保留字符来作为匹配字符 `{ } [ ] / \ + * . $ ^ | ?`。在特殊字符前面加 `\`，就可以使用它来做匹配字符。例如正则表达式 `.` 是用来匹配除了换行符以外的任意字符。现在要在输入字符串中匹配 `.` 字符，正则表达式 `(f|c|m)at\.?`，表示: 小写字母 `f`、`c` 或者 `m` 后跟小写字母 `a`，后跟小写字母 `t`，后跟可选的 `.` 字符。

```
"(f|c|m)at\.?" => The fat cat sat on the mat.
```

## 2.8 定位符

在正则表达式中，为了检查匹配符号是否是起始符号或结尾符号，我们使用定位符。定位符有两种类型: 第一种类型是 `^` 检查匹配字符是否是起始字符，第二种类型是 `$`，它检查匹配字符是否是输入字符串的最后一个字符。

### 2.8.1 插入符号

插入符号 `^` 符号用于检查匹配字符是否是输入字符串的第一个字符。如果我们使用正则表达式 `^a` (如果a是起始符号)匹配字符串 `abc`，它会匹配到 `a`。但是如果我们使用正则表达式 `^b`，它是匹配不到任何东西的，因为在字符串 `abc` 中 "b" 不是起始字符。让我们来看看另一个正则表达式 `^(T|t)he`，这表示: 大写字母 `T` 或小写字母 `t` 是输入字符串的起始符号，后面跟着小写字母 `h`，后跟小写字母 `e`。

```
"(T|t)he" => The car is parked in the garage.

"^(T|t)he" => The car is parked in the garage.
```

### 2.8.2 美元符号

美元 `$` 符号用于检查匹配字符是否是输入字符串的最后一个字符。例如正则表达式 `(at\.)$`，表示: 小写字母 `a`，后跟小写字母 `t`，后跟一个 `.` 字符，且这个匹配器必须是字符串的结尾。

```
"(at\.)" => The fat cat. sat. on the mat.

"(at\.)$" => The fat cat sat on the mat.
```

## 3. 简写字符集

正则表达式为常用的字符集和常用的正则表达式提供了简写。简写字符集如下:

| 简写 | 描述 |
| --- | --- |
| . | 匹配除换行符以外的任意字符 |
| \w | 匹配所有字母和数字的字符: `[a-zA-Z0-9_]` |
| \W | 匹配非字母和数字的字符: `[^\w]` |
| \d | 匹配数字: `[0-9]` |
| \D | 匹配非数字: `[^\d]` |
| \s | 匹配空格符: `[\t\n\f\r\p{Z}]` |
| \S | 匹配非空格符: `[^\s]` |

## 4. 断言

后行断言和先行断言有时候被称为断言，它们是特殊类型的 **非捕获组** (用于匹配模式，但不包括在匹配列表中)。当我们在一种特定模式之前或者之后有这种模式时，会优先使用断言。例如我们想获取输入字符串 `$4.44 and $10.88` 中带有前缀 `$` 的所有数字。我们可以使用这个正则表达式 `(?<=\$)[0-9\.]*`，表示: 获取包含 `.` 字符且前缀为 `$` 的所有数字。以下是正则表达式中使用的断言:

| 符号 | 描述 |
| --- | --- |
| ?= | 正向先行断言 |
| ?! | 负向先行断言 |
| ?<= | 正向后行断言 |
| ?<! | 负向后行断言 |

### 4.1 正向先行断言

正向先行断言认为第一部分的表达式必须是先行断言表达式。返回的匹配结果仅包含与第一部分表达式匹配的文本。要在一个括号内定义一个正向先行断言，在括号中问号和等号是这样使用的 `(?=...)`。先行断言表达式写在括号中的等号后面。例如正则表达式 `(T|t)he(?=\sfat)`，表示: 匹配大写字母 `T` 或小写字母 `t`，后面跟字母 `h`，后跟字母 `e`。在括号中，我们定义了正向先行断言，它会引导正则表达式引擎匹配 `The` 或 `the` 后面跟着 `fat`。

```
"(T|t)he(?=\sfat)" => The fat cat sat on the mat.
```

### 4.2 负向先行断言

当我们需要从输入字符串中获取不匹配表达式的内容时，使用负向先行断言。负向先行断言的定义跟我们定义的正向先行断言一样，唯一的区别是不是等号 `=`，我们使用否定符号 `!`，例如 `(?!...)`。我们来看看下面的正则表达式 `(T|t)he(?!\sfat)`，表示: 从输入字符串中获取全部 `The` 或者 `the` 且不匹配 `fat` 前面加上一个空格字符。

```
"(T|t)he(?!\sfat)" => The fat cat sat on the mat.
```

### 4.3 正向后行断言

正向后行断言是用于获取在特定模式之前的所有匹配内容。正向后行断言表示为 `(?<=...)`。例如正则表达式 `(?<=(T|t)he\s)(fat|mat)`，表示: 从输入字符串中获取在单词 `The` 或 `the` 之后的所有 `fat` 和 `mat` 单词。

```
"(?<=(T|t)he\s)(fat|mat)" => The fat cat sat on the mat.
```

### 4.4 负向后行断言

负向后行断言是用于获取不在特定模式之前的所有匹配的内容。负向后行断言表示为 `(?<!...)`。例如正则表达式 `(?<!(T|t)he\s)(cat)`，表示: 在输入字符中获取所有不在 `The` 或 `the` 之后的所有单词 `cat`。

```
"(?<!(T|t)he\s)(cat)" => The cat sat on cat.
```

## 5. 标记

标记也称为修饰符，因为它会修改正则表达式的输出。这些标志可以以任意顺序或组合使用，并且是正则表达式的一部分。

| 标记 | 描述 |
| --- | --- |
| i | 不区分大小写: 将匹配设置为不区分大小写。 |
| g | 全局搜索: 搜索整个输入字符串中的所有匹配。 |
| m | 多行匹配: 会匹配输入字符串每一行。 |

### 5.1 不区分大小写

`i` 修饰符用于执行不区分大小写匹配。例如正则表达式 `/The/gi`，表示: 大写字母 `T`，后跟小写字母 `h`，后跟字母 `e`。但是在正则匹配结束时 `i` 标记会告诉正则表达式引擎忽略这种情况。正如你所看到的，我们还使用了 `g` 标记，因为我们要在整个输入字符串中搜索匹配。

```
"The" => The fat cat sat on the mat.
```

```
"/The/gi" => The fat cat sat on the mat.
```

### 5.2 全局搜索

`g` 修饰符用于执行全局匹配 (会查找所有匹配，不会在查找到第一个匹配时就停止)。例如正则表达式 `/.(at)/g`，表示: 除换行符之外的任意字符，后跟小写字母 `a`，后跟小写字母 `t`。因为我们在正则表达式的末尾使用了 `g` 标记，它会从整个输入字符串中找到每个匹配项。

```
".(at)" => The fat cat sat on the mat.
```

```
"/.(at)/g" => The fat cat sat on the mat.
```

### 5.3 多行匹配

`m` 修饰符被用来执行多行的匹配。正如我们前面讨论过的 `(^, $)`，使用定位符来检查匹配字符是输入字符串开始或者结束。但是我们希望每一行都使用定位符，所以我们就使用 `m` 修饰符。例如正则表达式 `/at(.)?$/gm`，表示: 小写字母 `a`，后跟小写字母 `t`，匹配除了换行符以外任意字符零次或一次。而且因为 `m` 标记，现在正则表达式引擎匹配字符串中每一行的末尾。

```
"/.at(.)?$/" => The fat
                cat sat
                on the mat.
```

```
"/.at(.)?$/gm" => The fat
                  cat sat
                  on the mat.
```

## 常用正则表达式

- **正整数**: `^\d+$`
- **负整数**: `^-\d+$`
- **电话号码**: `^+?[\d\s]{3,}$`
- **电话代码**: `^+?[\d\s]+(?[\d\s]{10,}$`
- **整数**: `^-?\d+$`
- **用户名**: `^[\w\d_.]{4,16}$`
- **字母数字字符**: `^[a-zA-Z0-9]*$`
- **带空格的字母数字字符**: `^[a-zA-Z0-9 ]*$`

- **密码**: `^(?=^.{6,}$)((?=.*[A-Za-z0-9])(?=.*[A-Z])(?=.*[a-z]))^.*$`
- **电子邮件**: `^([a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4})*$`
- **IPv4 地址**: `^((?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))*$`
- **小写字母**: `^([a-z])*$`
- **大写字母**: `^([A-Z])*$`
- **网址**: `^(((http|https|ftp):\/\/)?([[a-zA-Z0-9]\-\.])+(\.)([[a-zA-Z0-9]]){2,4}([[a-zA-Z0-9]\/+=%&_\.~?\-]*))*$`
- **VISA 信用卡号码**: `^(4[0-9]{12}(?:[0-9]{3})?)*$`
- **日期** (MM/DD/YYYY): `^(0?[1-9]|1[012])[- /.](0?[1-9]|[12][0-9]|3[01])[- /.](19|20)?[0-9]{2}$`
- **日期** (YYYY/MM/DD): `^(19|20)?[0-9]{2}[- /.](0?[1-9]|1[012])[- /.](0?[1-9]|[12][0-9]|3[01])$`
- **万事达信用卡号码**: `^(5[1-5][0-9]{14})*$`