LEARN REGEX THE EASY WAY

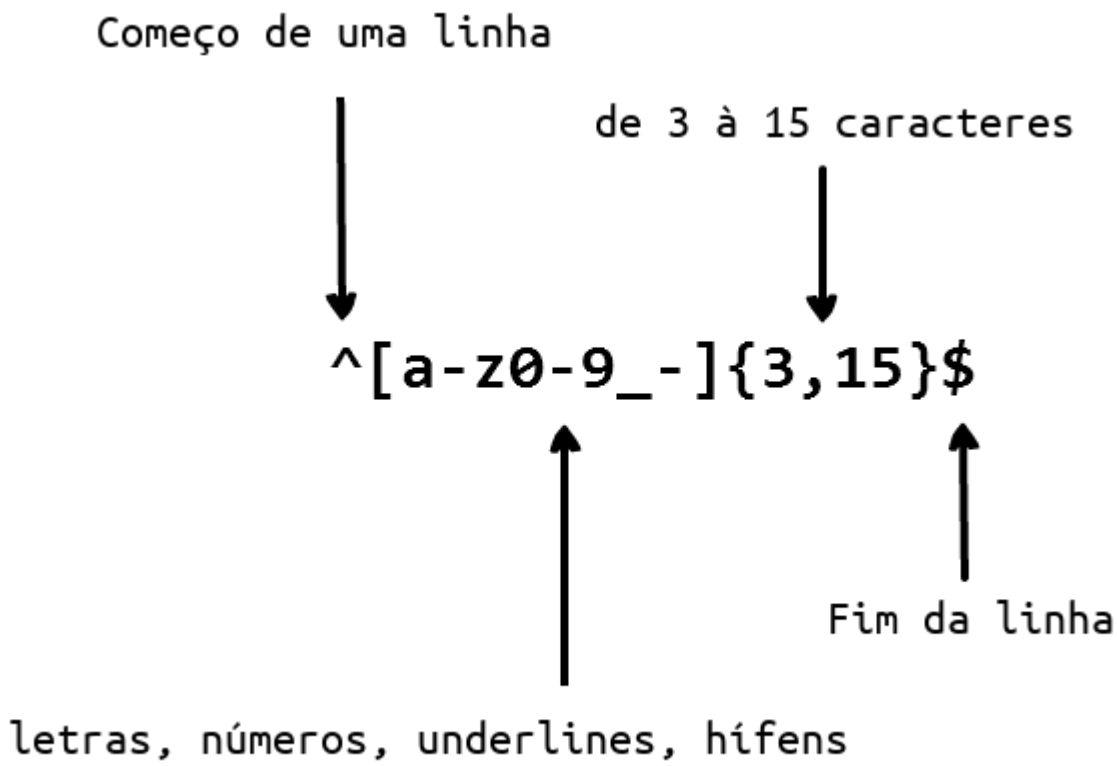
twitter tweet feedback @ziishaned

O que é uma Expressão Regular?

> Expressão Regular é um grupo de caracteres ou símbolos utilizado para encontrar um padrão específico a partir de um texto.

Uma expressão regular é um padrão que é comparado com uma cadeia de caracteres da esquerda para a direita. A expressão "Expressão regular" é longa e difícil de falar; você geralmente vai encontrar o termo abreviado como "regex" ou "regexp". Expressões regulares são usadas para substituir um texto dentro de uma string, validar formulários, extrair uma parte de uma string baseada em um padrão encontrado e muito mais.

Imagine que você está escrevendo uma aplicação e quer colocar regras para quando um usuário escolher seu username. Nós queremos permitir que o username contenha letras, números, underlines e hífens. Nós também queremos limitar o número de caracteres para não ficar muito feio. Então usamos a seguinte expressão regular para validar o username:



A expressão regular acima aceita as strings `john_doe`, `jo-hn_doe` e `john12_as`. Ela não aceita `Jo` porque essa string contém letras maiúsculas e também é muito curta.

Sumário

1. Combinações Básicas

Uma expressão regular é apenas um padrão de caracteres que usamos para fazer busca em um texto. Por exemplo, a expressão regular `the` significa: a letra `t`, seguida da letra `h`, seguida da letra `e`.

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

Teste a RegExp

A expressão regular `123` corresponde à string `123`. A expressão regular é comparada com uma string de entrada, comparando cada caractere da expressão regular para cada caractere da string de entrada, um após o outro. Expressões regulares são normalmente case-sensitive (sensíveis a maiúsculas), então a expressão regular `The` não vai bater com a string `the`.

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

Teste a RegExp

2. Metacaracteres

Metacaracteres são elementos fundamentais das expressões regulares. Metacaracteres não representam a si mesmos mas, ao invés disso, são interpretados de uma forma especial. Alguns metacaracteres tem um significado especial e são escritos dentro de colchetes. Os metacaracteres são os seguintes:

Metacaracter	Descrição
.	Corresponde a qualquer caractere, exceto uma quebra de linha
[ ]	Classe de caracteres. Corresponde a qualquer caractere contido dentro dos colchetes.
[^]	Classe de caracteres negada. Corresponde a qualquer caractere que não está contido dentro dos colchetes.
*	Corresponde a 0 ou mais repetições do símbolo anterior.
+	Corresponde a 1 ou mais repetições do símbolo anterior.
?	Faz com que o símbolo anterior seja opcional.
{n,m}	Chaves. Corresponde a no mínimo "n" mas não mais que "m" repetições do símbolo anterior.
(xyz)	Grupo de caracteres. Corresponde aos caracteres xyz nesta exata ordem.
|	Alternância. Corresponde aos caracteres antes ou os caracteres depois do símbolo
\	Escapa o próximo caractere. Isso permite você utilizar os caracteres reservados `[ ] ( ) { } . * + ? ^ $ \ |`
^	Corresponde ao início da entrada.
$	Corresponde ao final da entrada.

2.1 Ponto final

O ponto final `.` é um simples exemplo de metacaracteres. O metacaractere `.` corresponde a qualquer caractere sozinho. Ele não se iguala ao Enter e à quebra de linha. Por exemplo, a expressão regular `.ar` significa: qualquer caractere,

seguido da letra `a`, seguida da letra `r`.

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

Teste a RegExp

2.2 Conjunto de caracteres

Conjuntos de caracteres também são chamados de classes de caracteres. Utilizamos colchetes para especificar conjuntos de caracteres. Use um hífen dentro de um conjunto de caracteres para especificar o intervalo de caracteres. A ordem dos caracteres dentro dos colchetes não faz diferença. Por exemplo, a expressão regular `[Tt]he` significa: um caractere maiúsculo `T` ou minúsculo `t`, seguido da letra `h`, seguida da letra `e`.

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

Teste a RegExp

No entanto, um ponto final dentro de um conjunto de caracteres, significa apenas um ponto final. A expressão regular `ar[.]` significa: o caractere minúsculo `a`, seguido da letra `r`, seguida pelo caractere de ponto final `.`.

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

Teste a RegExp

2.2.1 Conjunto de caracteres negados

No geral, o símbolo do circunflexo representa o início da string, mas quando está logo após o colchete de abertura, ele faz a negação do conjunto de caracteres. Por exemplo, a expressão regular `[^c]ar` significa: qualquer caractere com exceção do `c`, seguido pelo caractere `a`, seguido da letra `r`.

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

Teste a RegExp

2.3 Repetições

Seguindo os metacaracteres `+`, `*` ou `?` são utilizados para especificar quantas vezes um sub-padrão pode ocorrer. Esses metacaracteres atuam de formas diferentes em diferentes situações.

2.3.1 O Asterisco

O símbolo `*` corresponde a zero ou mais repetições do padrão antecedente. A expressão regular `a*` significa: zero ou mais repetições do caractere minúsculo precedente `a`. Mas se o asterisco aparecer depois de um conjunto de caracteres, ou classe de caracteres, ele irá procurar as repetições de todo o conjunto. Por exemplo, a expressão regular `[a-z]*` significa: qualquer quantidade de letras minúsculas numa linha.

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

Teste a RegExp

O símbolo `*` pode ser usado junto do metacaractere `.` para encontrar qualquer string de caracteres `.*`. O símbolo `*` pode ser usado com o caractere de espaço em branco `\s` para encontrar uma string de caracteres em branco. Por exemplo, a expressão `\s*cat\s*` significa: zero ou mais espaços, seguidos do caractere minúsculo `c`, seguido do caractere minúsculo `a`, seguido do caractere minúsculo `t`, seguido de zero ou mais espaços.

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

Teste a RegExp

2.3.2 O Sinal de Adição

O símbolo `+` corresponde a uma ou mais repetições do caractere anterior. Por exemplo, a expressão regular `c.+t` significa: a letra minúscula `c`, seguida por pelo menos um caractere, seguido do caractere minúsculo `t`.

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

Teste a RegExp

2.3.3 O Ponto de Interrogação

Em expressões regulares, o metacaractere `?` faz o caractere anterior ser opcional. Esse símbolo corresponde a zero ou uma ocorrência do caractere anterior. Por exemplo, a expressão regular `[T]?he` significa: A letra maiúsculo `T` opcional, seguida do caractere minúsculo `h`, seguido do caractere minúsculo `e`.

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

Teste a RegExp

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

Teste a RegExp

2.4 Chaves

Em expressões regulares, chaves, que também são chamadas de quantificadores, são utilizadas para especificar o número de vezes que o caractere, ou um grupo de caracteres, pode se repetir. Por exemplo, a expressão regular `[0-9]{2,3}` significa: Encontre no mínimo 2 dígitos, mas não mais que 3 (caracteres no intervalo de 0 a 9).

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

Teste a RegExp

Nós podemos retirar o segundo número. Por exemplo, a expressão regular `[0-9]{2,}` significa: Encontre 2 ou mais dígitos. Se removermos a vírgula a expressão regular `[0-9]{3}` significa: Encontre exatamente 3 dígitos.

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

Teste a RegExp

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

Teste a RegExp

2.5 Grupo de Caracteres

Grupo de caracteres é um grupo de sub-padrão que é escrito dentro de parênteses `(...)`. Como falamos antes, se colocaramos um quantificador depois de um caractere, ele irá repetir o caractere anterior. Mas se colocarmos um quantificador depois de um grupo de caracteres, ele irá repetir todo o conjunto. Por exemplo, a expressão regular `(ab)*` corresponde a zero ou mais repetições dos caracteres "ab". Nós também podemos usar o metacaractere de alternância `|` dentro de um grupo de caracteres. Por exemplo, a expressão regular `(c|g|p)ar` significa: caractere minúsculo `c`, `g` ou `p`, seguido do caractere `a`, seguido do caractere `r`.

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

Teste a RegExp

2.6 Alternância

Em expressões regulares, a barra vertical `|` é usada para definir alternância. Alternância é como uma condição entre múltiplas expressões. Agora, você pode estar pensando que um conjunto de caracteres e a alternância funcionam da mesma forma. Mas a grande diferença entre eles é que o conjunto de caracteres trabalha no nível de caracteres, enquanto a alternância trabalha no nível das expressões. Por exemplo, a expressão regular `(T|t)he|car` significa: o caractere maiúsculo `T` ou minúsculo `t`, seguido do caractere minúsculo `h`, seguido do caractere minúsculo `e` ou o caractere minúsculo `c`, seguido do caractere minúsculo `a`, seguido do caractere minúsculo `r`.

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

Teste a RegExp

2.7 Escapando Caracteres Especiais

Em expressões regulares, a contrabarra `\` é usada para escapar o próximo caractere. Isso possibilita especificar um símbolo como um caractere correspondente, incluindo os caracteres reservados `{ } [ ] / \ + * . $ ^ | ?`. Para usar um caractere especial como um caractere correspondente, utilize `\` antes dele. Por exemplo, a expressão regular `.` é usada para encontrar qualquer caractere, exceto nova linha. Agora, para encontrar `.` em uma string de entrada, a expressão regular `(f|c|m)at\.?` significa: letra minúscula `f`, `c` ou `m`, seguida do caractere minúsculo `a`, seguido da letra minúscula `t`, seguida do caractere `.` opcional.

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

Teste a RegExp

2.8 Âncoras

Em empressões regulares, usamos âncoras para verificar se o caractere encontrado está no início ou no final da string de entrada. As âncoras podem ser de dois tipos: O primeiro tipo é o Acento Circunflexo `^`, que verifica se o caractere encontrado está no início da string de entrada, e o segundo tipo é o Sinal de Dólar `$`, que verifica se o caractere encontrado é o último caractere da string.

2.8.1 Acento Circunflexo

O símbolo do Acento Circunflexo `^` é usado para verificar se o caractere encontrado é o primeiro caractere da string de entrada. Se aplicarmos a seguinte expressão regular `^a` (se a é o primeiro caractere) à string de entrada `abc`, ela encontra o `a`. Mas se nós aplicarmos a expressão regular `^b` na mesma string, ela não encontrará nada. Isso acontece porque, na string `abc`, "b" não é o caractere inicial. Vamos dar uma olhada em outra expressão regular, `^(T|t)he` que significa: o caractere maiúsculo `T` ou o caractere minúsculo `t` que é o primeiro símbolo da string de entrada, seguido do caractere minúsculo `h`, seguido do caractere minúsculo `e`.

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

Teste a RegExp

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

Teste a RegExp

2.8.2 Sinal de Dólar

O símbolo do Sinal de Dólar `$` é usado para verificar se o caractere encontrado é o último caractere da string de entrada. Por exemplo, a expressão regular `(at\.)$` significa: um caractere minúsculo `a`, seguido do caractere minúsculo `t`, seguido de um ponto final `.` e o grupo deve estar no final da string.

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

Teste a RegExp

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

Teste a RegExp

3. Forma Abreviada de Conjunto de Caracteres

As expressões regulares fornecem abreviações para conjuntos de caracteres comumente usados, que oferecem atalhos convenientes para expressões regulares comumente usadas. As abreviações são as seguintes:

Abreviação	Descrição
.	Qualquer caractere, exceto nova linha
\w	Corresponde a caracteres alfanuméricos: `[a-zA-Z0-9_]`
\W	Corresponde a caracteres não alfanuméricos: `[^\w]`
\d	Corresponde a dígitos: `[0-9]`

Abreviação	Descrição
\D	Corresponde a não dígitos: `[^\d]`
\s	Corresponde a caracteres de espaços em branco: `[\t\n\f \r\p{Z}]`
\S	Corresponde a caracteres de espaços não em branco: `[^\s]`

4. Olhar ao Redor

Lookbehind (olhar atrás) e lookahead (olhar à frente), às vezes conhecidos como lookarounds (olhar ao redor), são tipos específicos de ***grupo de não captura*** (utilizado para encontrar um padrão, mas não incluí-lo na lista de ocorrêncoas). Lookarounds são usados quando temos a condição de que determinado padrão seja precedido ou seguido de outro padrão. Por exemplo, queremos capturar todos os números precedidos do caractere `$` da seguinte string de entrada: `$4.44 and $10.88`. Vamos usar a seguinte expressão regular `(?<=\$)[0-9\.]*` que significa: procure todos os números que contêm o caractere `.` e são precedidos pelo caractere `$`. A seguir estão os lookarounds que são utilizados em expressões regulares:

Símbolo	Descrição
?=	Lookahead Positivo
?!	Lookahead Negativo
?<=	Lookbehind Positivo
?<!	Lookbehind Negativo

4.1 Lookahead Positivo

O lookahead positivo impõe que a primeira parte da expressão deve ser seguida pela expressão lookahead. A combinação retornada contém apenas o texto que encontrado pela primeira parte da expressão. Para definir um lookahead positivo, deve-se usar parênteses. Dentro desses parênteses, é usado um ponto de interrogação seguido de um sinal de igual, dessa forma: `(?=...)`. Expressões lookahead são escritas depois do sinal de igual dentro do parênteses. Por exemplo, a expressão regular `[T|t]he(?=\sfat)` significa:

encontre a letra minúscula `t` ou a letra maiúscula `T`, seguida da letra `h`, seguida da letra `e`. Entre parênteses, nós definimos o lookahead positivo que diz para o motor de expressões regulares para encontrar `The` ou `the` que são seguidos pela palavra `fat`.

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

Teste a RegExp

4.2 Lookahead Negativo

O lookahead negativo é usado quando nós precisamos encontrar todas as ocorrências da string de entrada que não são seguidas por um determinado padrão. O lookahead negativo é definido da mesma forma que definimos o lookahead positivo, mas a única diferença é que, no lugar do sinal de igual `=`, usamos o caractere de negação `!`, ex.: `(?!...)`. Vamos dar uma olhada na seguinte expressão regular `[T|t]he(?!\sfat)`, que significa: obtenha as palavras `The` ou `the` da string de entrada que não são seguidas pela palavra `fat`, precedida de um caractere de espaço.

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

Teste a RegExp

4.3 Lookbehind Positivo

Lookbehind positivo é usado para encontrar todas as ocorrências que são precedidas por um padrão específico. O lookbehind positivo é indicado por `(?<=...)`. Por exemplo, a expressão regular `(?<=[T|t]he\s)(fat|mat)` significa: obtenha todas as palavras `fat` ou `mat` da string de entrada, que estão depois das palavras `The` ou `the`.

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

Teste a RegExp

4.4 Lookbehind Negativo

Lookbehind negativo é usado para encontrar todas as ocorrências que não são precedidas por um padrão específico. O lookbehind negativo é indicado por `(?<!...)`. Por exemplo, a expressão regular `(?<!(T|t)he\s)(cat)` significa: obtenha todas as palavras `cat` da string de entrada, que não estão depois das palavras `The` ou `the`.

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

Teste a RegExp

5. Flags

Flags (sinalizadores) também são chamados de modificadores, porque eles modificam o resultado da expressão regular. Essas flags podem ser usadas em qualquer ordem ou combinação, e são uma parte integrante da RegExp.

Flag	Descrição
i	Case insensitive: Define que o padrão será case-insensitive.
g	Busca global: Procura o padrão em toda a string de entrada.
m	Multilinhas: Os metacaracteres de âncora funcionam em cada linha.

5.1 Indiferente a Maiúsculas

O modificador `i` é usado para tornar o padrão case-insensitive. Por exemplo, a expressão regular `/The/gi` significa: a letra maiúscula `T`, seguida do caractere minúsculo `h`, seguido do caractere `e`. E ao final da expressão regular, a flag `i` diz ao motor de expressões regulares para ignorar maiúsculas e minúsculas. Como você pode ver, nós também determinamos a flag `g` porque queremos procurar o padrão em toda a string de entrada.

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

Teste a RegExp

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

Teste a RegExp

5.2 Busca Global

O modificador `g` é usado para realizar uma busca global (encontrar todas as ocorrências sem parar na primeira encontrada). Por exemplo, a expressão regular `/.(at)/g` significa: qualquer caractere, exceto nova linha, seguido do caractere minúsculo `a`, seguido do caractere minúsculo `t`. Por causa da flag `g` no final da expressão regular, agora ela vai encontrar todas as ocorrências em toda a string de entrada.

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

Teste a RegExp

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

Teste a RegExp

5.3 Multilinhas

O modificador `m` é usado para realizar uma busca em várias linhas. Como falamos antes, as âncoras `(^, $)` são usadas para verificar se o padrão está no início ou no final da string de entrada. Mas se queremos que as âncoras funcionem em cada uma das linhas, usamos a flag `m`. Por exemplo, a expressão regular `/at(.)?$/gm` significa: o caractere minúsculo `a`, seguido do caractere minúsculo `t`, opcionalmente seguido por qualquer caractere, exceto nova linha. E por causa da flag `m`, agora o motor de expressões regulares encontra o padrão no final de cada uma das linhas da string.

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

Teste a RegExp

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

```
                    on the mat.
```

Teste a RegExp

Contribution

- Reporte bugs
- Abra pull request com melhorias
- Espalhe a palavra
- Me encontre diretamente em ziishaned@gmail.com ou Follow @ziishaned

Licença

MIT © Zeeshan Ahmad