#### 

#### 

#### SUPERIOR EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Sistemas de Inovação e Empreendedorismo

Danillo de Souza Koch

EXPRESSÕES REGULARES

#### JOINVILLE-SC

2025

1. **Definição**

**O que são Expressões Regulares?**

Expressões regulares, conhecidas como regex, são uma linguagem para descrever padrões em strings. Elas formam uma linguagem pequena e separada, incorporada ao JavaScript (e em várias outras linguagens de programação, de uma forma ou de outra). Também oferecem uma maneira flexível de pesquisar, manipular texto e validar dados.

Não é uma linguagem muito legível — expressões regulares grandes tendem a ser completamente ilegíveis. Mas é uma ferramenta útil e pode realmente simplificar programas de processamento de strings.

Em outras palavras, Expressões regulares são uma forma de descrever padrões em uma sequência de [dados](https://builtin.com/software-engineering-perspectives/javascript-data-structures) . Elas formam uma pequena linguagem própria, que faz parte de muitas linguagens de programação como [JavaScript](https://builtin.com/software-engineering-perspectives/javascript) , Perl, [Python](https://builtin.com/software-engineering-perspectives/python) , [PHP](https://builtin.com/software-engineering-perspectives/php) e [Java](https://builtin.com/software-engineering-perspectives/java) .

1. **Como Usar:**

**Metacaracteres**

A maioria dos caracteres são literais, ou seja, casam somente com eles próprios, por exemplo **t** teste "t", as exceções são os metacaracteres

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

Cada metacaractere representa um grupo distinto sendo elas: Classes, Quantificadores, Agrupamento, Ancoras e outros

**Classes**

São definidas entre colchetes [], isso indica que estamos buscando especificamente o caractere informado dentro dos colchetes

por exemplo, dado determinado texto quero que encontre a letra "a" então escrevemos o seguinte padrão [a]

existem várias classe de caracteres prontas que podemos utilizá-las para facilitar nossas operações

estas são algumas classes comumente utilizadas

**[A-Z]** - significa de A até Z, sempre maiúscula.

**[a-z]**- significa de a até z, sempre minúscula,

**[A-Za-z]**- significa A-Z ou a-z.

**\d** - significa todos os dígitos e é atalho para [0-9]

**\w**- significa word char e é um atalho para [A-Za-z0-9\_].

**\s**significa whitespace normalmente é um atalho para [ \t\r\n\f] onde:

O primeiro caractere é um espaço branco.

**\t** é um tab.

**\r** é carriage return.

**\n** é newline.

**\f** é form feed.

Quantificadores

São identificadores para dizer quantas vezes determinada expressão deve aparecer em sequência

**?** - zero ou uma vez.

**\***- zero ou mais vezes.

**+** - uma ou mais vezes.

**.** - o "ponto" que significa qualquer caractere

**{** e **}** - as chaves servem para definir uma quantidade de caracteres específicas que é desejado encontrar

 Por exemplo:

**a{3}** - letra **a** 3 vezes.

**{**n**}** - exatamente n vezes.

**{**n,**}** - no mínimo n vezes.

**{**n,m**}** - no mínimo n vezes, no máximo m vezes.

Com \ (barra) podemos escapar metacaracteres, se quisermos procurar pelo \* ou . literalmente (sem significado especial), devemos utilizar o caractere \

Por exemplo: \..

Podemos combinar classes com quantificadores o que nos dá ainda mais poder pesquisar informações, alguns exemplos:

**\d\***- um dígito, zero ou mais vezes

**[.-]\***   - ponto ou hífen zero, uma ou mais vezes.

**[.-]{1}** - ponto ou hífen uma vez.

**[.-]?**  - ponto ou hífen zero ou uma vez.

**Agrupamentos**

São representados pelos metacaracteres **( )**, utilizamos para agrupar regras ou extrair informações

**()** - Declaramos um grupo

**?:** - Non-Capturing group

exemplo **(?**:de\s+**)**  não deve devolver o grupo formado pela preposição de e por um \s

**Âncoras**

Quando precisamos encontrar determinada informação delimitando uma parte da informação podemos usar as âncoras:

**\b** - (minúsculo) é uma âncora que seleciona um word boundary, isso é o início ou fim da palavra.

**^** - Garantir que nada vem antes.

**\. –** Transforma caracteres especiais em literais como +-\_ normalmente utilizado em e-mail.

**$**- Garantir que nada vem depois.

**\B** - (B maiúsculo) non-word-boundary comportamento inverso a sílaba alvo deve aparecer dentro de uma palavra, nunca no início ou fim.

**Negação**

Às vezes é mais vantajoso selecionarmos a informação que não queremos e para isso podemos utilizar

**\W**  - (maiúsculo) é a non-word char, ou seja, tudo que não é um word char. \W é um atalho para**[^\w]**.

**\D** - (maiúsculo) é um non-digit, sendo \D um atalho para **[^\d]**

Repare também que não usamos a metacaractere **^** como âncora, pois aparece dentro de uma classe**[^>]**

Com esses conceitos estabelecidos já é possível criar um regex para uma finalidade específica ou então compreender um padrão já existente que precise ser modificado

1. ***COMO APLICAR NO JAVASCRIPT***

Vamos analisar um exemplo do cotidiano que utilizamos regex

regex para validação de formatação de CPF:

**(\d{3})[.]?(\d{3})[.]?(\d{3})[-]?(\d{2})**

Separando em cores podemos perceber que existem um conjunto de regras sendo aplicadas com o objetivo de validar se a informação está no formato correto de um CPF

Vamos descrever a regras e exibir a parte correspondente desta regra

CPF é composto  por

* 3 dígitos e após os dígitos pode ou não estar separados por um ponto

**(\d{3})[.]?**

* Seguido de mais 3 dígitos e após os dígitos pode ou não estar separados por um ponto

**(\d{3})[.]?**

* Seguido de mais 3 dígitos e após os dígitos pode ou não estar separado por um traço

**(\d{3})[-]?**

* Seguido de mais 2 dígitos

**(\d{2})**

Agora, vamos analisar mais um regex **^([\w]\.?)+@([\w]+\.)+([a-z]{2,4})+$**que é bastante utilizado e a princípio parece bem complexo, mas quando separamos cada parte da regra, quebramos essa complexidade e fica mais simples de entender qual o objetivo desta expressão como um todo

Vamos separar visualmente em cores para facilitar a compreensão

**^([\w]\.?)+@([\w]+\.)+([a-z]{2,4})+$**

Agora vamos descrever cada regra pelas suas cores

* âncora que define que nada deve vir antes
  + agrupe
    - Classe de caracteres (letras e números)
    - Caractere literal ponto pode aparecer 0 ou 1 vez
  + deve aparecer uma ou mais vezes
* deve conter o caractere literal @
  + agrupe
    - Classe de caracteres (letras e números) 1 ou mais vezes
    - caractere literal ponto pode aparecer 0 ou 1 vez
  + deve aparecer uma ou mais vezes
  + agrupe
    - Classe de caracteres (apenas letras minúsculas)
      * no mínimo 2 e no máximo 4 caracteres
  + deve aparecer uma ou mais vezes
  + âncora que define que nada deve vir depois

**REFERÊNCIAS**

Haverbeke, M. \*Eloquent JavaScript: A Modern Introduction to Programming\*. 1. ed. Disponível em: https://eloquentjavascript.net/1st\_edition/chapter10.html. Acesso em: 04 set. 2025.

ROCKETSEAT. Expressões Regulares em JavaScript. Disponível em: https://www.rocketseat.com.br/blog/artigos/post/javascript-expressoes-regulares. Acesso em: 04 set. 2025.

BUILTIN. JavaScript Regex Guide: How to Use Regular Expressions. Disponível em: https://builtin.com/software-engineering-perspectives/javascript-regex. Acesso em: 04 set. 2025.