Expressões regulares: Capturando textos de forma mágica (regex)

- Regex, ou expressões regulares, é uma linguagem para encontrar padrões de texto.
- Uma expressão regular sozinha é apenas uma string. É preciso ter um software para interpretar a regex e aplicá-la no alvo. Esse software é o Regex Engine que existe para a maioria das plataformas de desenvolvimento, como JavaScript, C#, Python, Ruby ou PHP.
- Explicação: Existem alguns caracteres que possuem um significado especial para o regex engine. Especial significa que o regex engine não interpreta o valor literal e sim diferente. Esses caracteres são chamados de **meta caracteres**.

Nessa aula já vimos alguns:

- o "ponto" que significa qualquer char
- * o asterisco que serve para definir uma quantidade de chars, zero ou mais vezes
- { e } as chaves que servem para definir uma quantidade de caracteres específicas que é desejado encontrar

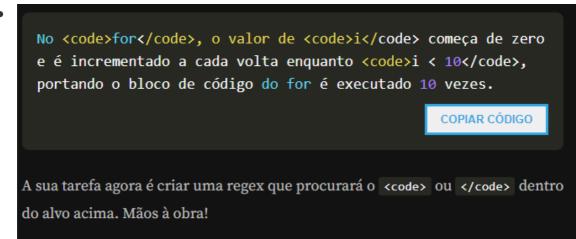
Por exemplo:

- a{3} letra a 3 vezes.
- \d* um digito zero ou mais vezes

Lembrando também, se quisermos procurar pelo * ou . literalmente (sem significado especial), devemos utilizar o caractere \

- Uma expressão regular faz a análise de um determinado padrão de caracteres em uma string. Podemos usar esse padrão para por exemplo validar um email ou telefone.
- ? é um quantifier que deixa como opção, por exemplo o . no cpf pode ser opcional, entao é \d{3}\.?

 [] -> define uma classe de caracteres possíveis que pode ter, ou seja, se for possível colocar . ou - antes dos ultimos 2 digitos do cpf, deve ser feito da seguinte maneira: [-.]



- Resposta: </?code>
- Para pegar espaço: \s
- 1 ou mais vezes: +
- 0 ou mais vezes: *
 - Podemos definir facilmente a classe de qualquer caractere com o [A-Z].
 - Conhecemos todos os quantifiers como ?, +, * e {n}.
 - \s significa whitespace e é um atalho para [\t\r\n\f].
 - \w significa word char e é uma atalho para [A-Za-z0-9_].
- ^ é início
- \$ é final
- (?:) -> esse grupo não vai ser colocado no resultado da execução
- Greedy or Lazy mode: metachar
- Como assim? Nossa regex é gananciosa por padrão e selecionou todos os caracteres até o último > . O meta-char, que na verdade é ganancioso, é o + , igualmente * e {} são também assim, e sempre selecionam o máximo de caracteres possíveis, se não for configurado diferente. Ou seja, podemos dizer que não queremos "ganância" e sim preguiçoso ou hesitante. Isso se faz, novamente pelo caractere ? :

 <a href="https://documents.org/linearin

• Todos os quantifiers são gananciosos por padrão. Isso significa que eles automaticamente selecionam o máximo de caracteres por padrão.

- backreferences: sintaxe é \1 , onde 1 é o número do grupo referenciado (usado para lidar, por exemplo, com tags html que abrem e fecham iguais)
- Há uma alternativa para resolver esse problema de abertura da tag. Podemos definir uma classe de caracteres que seleciona tudo que não é um >. Essa negação é feita através da meta-char ^
 - Essa negação é algo muito comum nas regexes. Há circunstâncias em que é mais fácil definir o que **não queremos** em vez de dizer o que queremos. A negação ^ ajuda nisso. Isso também é a razão da existência de classes como w (com W maiúsculo) e v (com D maiúsculo).

A classe \D , por sua vez, é um *non-digit*, ou seja, \D é um atalho para [^\d]

Repare também que **não usamos a meta-char** ^ **como âncora** pois aparece dentro de uma classe [^>] .

•

Dúvidas e assuntos a pesquisar

• regex101: build, test, and debug regex