A letra g é uma flag do mundo JavaScript e significa global, para aplicar a regex na string inteira (e não parar no primeiro match)

var objetoRegex = new RegExp(textoPattern, 'g');

. 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

\d - representa um numero

Exemplo: expressão regular para cpf

\d{3}\.\d{3}\.\d{3}-\d{2}

Caso haja o tamanho esteja em um intervalo, usar a expressão abaixo:

\d{1,3} - pode ter de 1 a 3 caracteres

O ? segnifica que pode ou não ter o caracter, no exmeplo abaixo não é obrigatório o ‘-’

\d{5}-?\d{3}

Também pode colocar o \d{0,1}

Para definir quando pode aparecer um ou outro caracter usa-se[ ], o \* é para definir a quatidade de vezes(0 ou 1 vez)

\d{5}**[-.]\***\d{3}

Definir de qual numero até qual numero deve aparecer

[0-9]

Classe para dois intervalos de numeros

[1-36-9]

significa whitespace e é um atalho para [ \t\r\n\f], quando não coloco numero final na quatidade é considerado ilimitado

\s{1,}

+ - uma ou mais vezes

Exemplo pegar data por extenso

[1-3]?\d\s+de\s+[A-Z][a-zç]{1,8}\s+de\s+\d{4}

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

\bde\b- pega exatamente o que foi solicitado,. nesse caso o ‘de’. Para funcionar não pode ter um word char seguido[A-Za-z0-9\_]

\Bpor\B - é o inverso, quando queremos que tenha um \w nas laterais

Quando queremos algo do inicio do alvo é usado o ^ e para o final o $, segue exemplo

Alvo: file:///Users/nico/Downloads/regex/index.html

Expressão: ^file.+\.html$

Para definir um grupo é colocado entre ()

Para não devolver um grupo pela regex - non-capturing groups

(?:expressão\_normal)

O pipe ‘|’ serve como operador lógico ou

Expressões gananciosas

No alvo abaixo:

<h1 class="text-left">Expressões regulares</h1>

Ao aplicar a expressão: <h1.+?>, ira ser retornado até o ultimo >, não parando no primeiro

Para corrigir isso e deixar a expressão preguiçosa, usamos o ? :

Exemplo: <h1.+?>

Outra alterativa seria criar uma classe com o ^, como no exemplo abaixo, Onde vai see=lecionar tudo o que não for ‘>’

<h1[^>]+>

No caso do elemento h1, no final deve ter tbm um h1, para fazer essa referencia os grupos são numerados, no exemplo abaixo o grupo de referencia é o 1

<(h1|h2).+?>([\w\sõãí.]+)<\/\1>

Alvo: Z171PZ7AZ23PZ7819AZ78GZ1AZ99IZ34O

Expressão: [^Z\d]

Nesse exemplo acima, não é mostrado as regras que constam dentro das chaves

**Exemplo aplicado a linguagens de programação**

Javascript

O exec, serve para executar a expressão no alvo definido, onde vai retornar um array com o resultado. Para saber se a expressão ira retornar verdadeiro ou falso para a expressão é usado a função .test(lavo)

var alvo = '11a22b33c';

var exp = /(\d\d)(\w)/g;

console.log(exp.test(alvo)); //true

while(resultado = exp.exec(alvo)){

console.log(resultado);

console.log(exp.lastIndex);

}

Outro exemplo

var alvo = '2007-12-31';

var exp = /-/g;

anoMesDia = alvo.replace(exp, '/');

console.log(anoMesDia);

Outro exemplo, Toda string em JavaScript possui o método match. Ele aceita uma expressão regular e retorna em um array todas as partes do alvo que atendem a expressão regular. Usando match

var codigos = 'A121B12112C12212F12G01';

var exp = /[A-Za-z]\d+/g

var codigosExtraidos = codigos.match(exp);

console.log(codigosExtraidos);

As funções replace e Split do js tbm aceitam expressões regulares

O HTML5 introduziu para as tags input o atributo pattern. Ele recebe como atributo expressões regulares, inclusive, quando o formulário for submetido e o valor digitado pelo usuário não for compatível com a expressão, o próprio navegador exibirá automaticamente uma mensagem para o usuário indicando que o campo é inválido. Mas é importante que o input faça parte de um formulário, caso contrário a validação não será aplicada.

<form>

<input pattern="[0-9]\*">

<input type="submit" value="Enviar dados">

</form>

Com PHP

Retorna um boleando, indicando se achou algum grupo

$regex = '~(\d\d)(\w)~';

$alvo = '11a22b33c';

$resultado = preg\_match($regex, $alvo, $match);

Os resultados encontrados, ficam armazenados no ultimo parâmetro $match, o resultado é impresso separando cada validação por grupos

Preg\_match\_all retorna a quantidade de todos os grupos

Exemplo para trocar formatação de data

$string = 'Setembro 21';

$regex = '~(\w+)\s(\d+)~';

$novoTexto = '$2 de $1';

$resultado = preg\_replace($regex, $novoTexto, $string);

echo $resultado; // 21 de Setembro