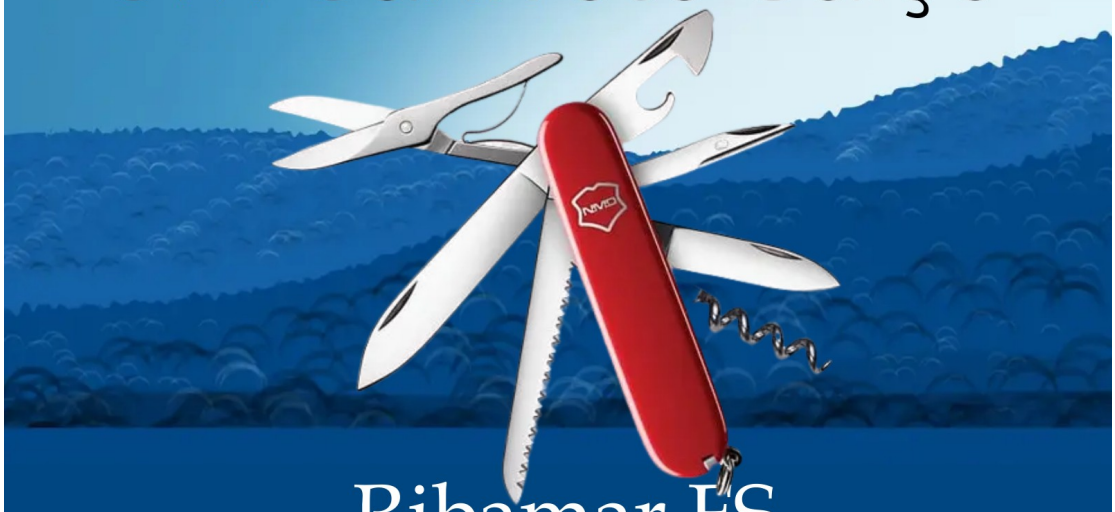
Expressões Regulares

(RegEx)

Um Canivete Suiço



Ribamar FS

Abril/2024

https://ribafs2.github.io/portal/

Sumário

Começando

O objetivo principal da elaboração deste pequeno e-book é fornecer, primeiro para mim (para me auxiliar em meus projetos), e depois de testado algumas vezes, também para outras pessoas interessadas em usar expressões regulares em seus projetos. Além do e-book, como PDF não guarda bem código a ser copiado e colado, então criei um repositório no Github. (https://github.com/ribafs2/regex) para guardar todo o código do e-book.
Aqui não tem muita informação sobre as expressões regulares, mas o pouco que trouse tem a pretensão d estimular você a aprender a usar este verdadeiro canivete suiço a seu favor.

Público Alvo

Pra valer, público-alvo é qualquer pessoa interessada, pois está disponível para todos.

Sobre o Autor

Sou o ribafs/Ribamar FS, um apaixonado pela programação web com PHP, por Linux e pela administração de servidores linux.

Participo ativamente em diversos grupos de discussão no Facebook

Estudo e trabalho com TI há uns 30 anos.

Um currículo não atualizado
https://ribafs.github.io/sobre/curriculo/

Outros livros meus (todos free)
https://ribafs.github.io/sobre/livros/

Planejamento

Ajuda muito que ao iniciar qualquer projeto façamos um planejamento do mesmo. Elaborando um roteiro, detalhando o que conterá o projeto e mais detalhes que considerarmos importantes.

Este planejamento deve ser mais elaborado sempre que o projeto for mais importante.

1 – Introdução

As expressões regulares são um verdadeiro canivete suíço para o programador, pois podem resolver alguns problemas inclusive complexos com grande facilidade caso saibamos como usar.

Como seu uso não é intuitivo, elaborei este pequeno livro.

Expressões Regulares é uma forma avançada de mexer com strings. Elas permitem que você crie condições para que um trecho seja encontrado, substituído ou uma string separada em uma array.

Uma expressão regular, na informática, define um padrão a ser usado para procurar ou substituir palavras ou grupos de palavras. É um meio preciso de se fazer buscas de determinadas porções de texto.

Por exemplo, se o conjunto de palavras for {asa, carro, jardim, ovos, terra} e a expressão regular buscar pelo padrão **rr**, obterá as palavras **carro e terra**.

A utilidade do RegExp é apenas limitada pela sua imaginação e conhecimento.

As ERs são úteis para buscar ou validar textos variáveis como:

- data
- horário
- número IP
- endereço de e-mail
- endereço de Internet (url)
- declaração de uma função()
- dados na coluna N de um texto
- dados que estão entre tags /tags
- número de telefone, RG, CPF, cartão de crédito
- entre outros...

Uma expressão regular é uma notação para representar padrões em strings. Serve para validar entradas de dados ou fazer busca e extração de informações em textos/strings.

Por exemplo, para verificar se um dado fornecido é um número de 0,00 a 9,99 pode-se usar a expressão regular \d,\d\d, pois o símbolo \d é um coringa que casa com um dígito/algerismo.

O verbo casar aqui está sendo usado como tradução para match, no sentido de combinar, encaixar, parear. Dizemos que a expressão \d,\d\d casa com 1,23 mas não casa com 123 (falta a vírgula) nem com 1,2c (“c” não casa com \d, porque não é um dígito).

Qualquer solução de descoberta e classificação de dados depende bastante de expressões regulares

Expressões regulares diferenciam maiúsculas e minúsculas

Este conceito ou ferramenta (regex) é usado em quase todas as linguagens de programação ou script, como PHP, C, C++, Java, Perl, JavaScript, Python, Ruby e muitas outras.

Também é usado em editores e processadores de texto como o word, que auxilia os usuários na busca do texto em um documento, e também em diversos IDEs.

O padrão definido pela expressão regular é aplicado a uma determinada string ou texto da esquerda para a direita.

2 – Conceitos

[a-z] - qualquer letra minúslula atende
[A-Z] - qualquer letra maiúscula atende
[0-9] - qualquer algarismo, de 0 a 9 atende
[a-zA-Z] - Um caractere maiúsculo ou minúsculo
[a-zA-Z0-9] - qualquer letra ou algarismo
[a-z0-9]
[0-9] - Um algarismo de 0 a 9. ou [0123456789]
[0-9](3) - 3 algarismos de 0 a 9
\w - uma letra w é de word (perceba que é w minúsculo)
\W - qualquer coisa menos letra (perceba que é W maiúsculo)
\s - espaço
\S - qualquer coisa menos espaço
\d - um digito
\D - qualquer coisa menos um digito
\ Escapa o próximo caractere, utilizado para procurar uma / por exemplo
^ Linha começando em
. Qualquer caractere exceto nova linha
$ Final da linha
| Ou
i - ignore case
(.+) - alguma coisa necessariamente com no mínimo 1 caractede
(.*) - pode ter alguma coisa ou pode não ter
[:lower:]: qualquer letra minúscula
[:upper:]: qualquer letra maiúscula
[:alpha:]: qualquer letra
[:alnum:]: qualquer letra ou dígito
[:digit:]: qualquer dígito/algarismo
\w: qualquer caractere para montar palavras
\s: qualquer caractere de espaço ou quebra de linha

Semelhantes
[A-Za-z0-9] ~ [[:upper:][:lower:][:digit:]] ~ [[:alpha:][:digit:]] ~ [[:alnum:]]
Ou - |
Grupos de caracteres - [at]
As expressões regulares/regex são suportadas nas principais linguagens de pprogramação, nos editores de texto, processadores de texto e IDEs.

Operadores Comuns da Regex

Operador	**Propósito**
. (ponto)	Corresponder a qualquer caractere único
A	Corresponder a uma letra maiúscula A
a	Corresponder a uma letra minúscula a
\d	Corresponder a qualquer dígito único
\D	Corresponder a qualquer caractere não dígito

[A-E] Corresponder a qualquer letra maiúscula A, B, C, D ou E
[^A-E] Corresponder a qualquer caractere, exceto à letra maiúscula A, B, C, D ou E
X? Corresponder a nenhuma ou a uma letra maiúscula X
X* Corresponder a zero ou mais letras maiúsculas X
X+ Corresponder a uma ou mais letras maiúsculas X
X{n} Corresponder exatamente a n letras maiúsculas X
X{n,m} Corresponder a pelo menos n e não mais do que m letras maiúsculas X ; se você omitir m, a expressão tenta corresponder pelo menos nX
(abc|def)+ Corresponder uma sequência de pelo menos um abc e def;abc e def corresponderiam

Metacaracteres
Os caracteres a seguir não são interpretados como literais e tem significados especiais
- • . ^ $ * + ? { } [ ] \ | ( )

Embora o Unix e o Linux os tenham tornado populares, as expressões regulares estão disponíveis em uma variedade de pacotes, incluindo o Microsoft Word.
As expressões regulares são usadas principalmente em vários programas Linux notáveis, incluindo grep, Awk e Sed.

OS METACARACTERES

Para saber como funcionam as Expressões Regulares precisamos primeiro conhecer os metacaracteres. Cada metacaractere é uma ferramenta que tem uma função específica. Eles servem para dar mais poder às pesquisas, formando padrões e posições impossíveis de se especificar usando somente caracteres normais.

Os metacaracteres são pequenos pedacinhos simples que agrupados entre si, ou com caracteres normais, formam algo maior, uma expressão. O importante é compreender bem cada um individualmente, e depois apenas lê−los em seqüência.

Para matar a curiosidade, aqui está os tão falados metacaracteres: . ? * + ^ $ | [ ] { } ( ) . Temos que nos acostumar com estes símbolos e seus respectivos nomes, então vai uma tabela com a meta e o mnemônico (nome).

meta	**mnemônico**
.	ponto
[]	lista
[^]	lista negada
?	opcional
*	asterisco
+	mais
{}	chaves
^	circunflexo
$	cifrão
b	borda
	escape
|	ou
()	grupo
1	retrovisor

REPRESENTANTES

meta	mnemônico	- função
.	- ponto	um caractere qualquer
[...]	lista	lista de caracteres permitidos
[^...]	lista negada	- lista de caracteres proibidos

QUANTIFICADORES

meta	mnemônico	- função
?	- opcional	zero ou um
*	- asterisco	- zero, um ou mais
+	- mais	- um ou mais
{n,m}	chaves	de n até m

ÂNCORAS

meta	mnemônico	- função
b	- borda	início ou fim de palavra

OUTROS

meta	mnemônico	- função
c	- escape	torna literal o caractere c
|	- ou	- ou um ou outro
(...)	grupo	delimita um grupo
1...9	retrovisor	- texto casado nos grupos 1...9

3 – Aplicação Prática

Exemplos práticos de uuso das regex

Telefone

```
$tel = '(85)-99440-7071';       //85994407071 ou (85)99440-7071 ou (85)-99440-7071
if(preg_match("/\(?\d{2}\)-?\s?\d{5}\-?\d{4}/", $tel)) {
  print "o telefone é válido";
}else{
  print "erro";
}
```

CEP

```
/^[0-9]{5}\-[0-9]{3}$/
```

CPF

/^[0-9]{3}.[0-9]{3}.[0-9]{3}-[0-9]{2}$/

Sem nenhuma máscara

/^[0-9]{11}$/

Data

/^[0-9]{2}\/[0-9]{2}\/[0-9]{4}$/

MasterCard

^(?:5[1-5][0-9]{2}|222[1-9]|22[3-9][0-9]|2[3-6][0-9]{2}|27[01][0-9]|2720)[0-9]{12}$

Visa

\b([4]\d{3}[\s]\d{4}[\s]\d{4}[\s]\d{4}|[4]\d{3}[-]\d{4}[-]\d{4}[-
]\d{4}|[4]\d{3}[.]\d{4}[.]\d{4}[.]\d{4}|[4]\d{3}\d{4}\d{4}\d{4})\b

Tags HTML

<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)

Hexadecimal

\B#(?:[a-fA-F0–9]{6}|[a-fA-F0–9]{3})\b

E-mail

\b[\w.!#$%&’*+\/=?^`{|}~-]+@[\w-]+(?:\.[\w-]+)*\b

IPv4

\b(?:(?:25[0-5]|2[0-4]\d|[01]?\d\d?)\.){3}(?:25[0-5]|2[0-4]\d|[01]?\d\d?)\b

Telefone

(99) [0-9]{4}[0-9]{3}[0-9]{3}

E-mail
^[\w\.=-]+@[\w\.-]+\.[\w]{2,3}$

CPF
/(?!(\d)\1{2}.\1{3}.\1{3}-\1{2})\d{3}\.\d{3}\.\d{3}\-\d{2}/gm

Data - dd/mm/yyyy
^([1][12]|[0]?[1-9])[\/-]([3][01]|[12]\d|[0]?[1-9])[\/-](\d{4}|\d{2})$

Extra
Máscara

```
function format_string($mask, $str, $ch = '#') {
    $c = 0;
    $rs = '';

    // Aqui usamos strlen() pois não há preocupação com o charset da máscara.
    for ($i = 0; $i < strlen($mask); $i++) {
        if ($mask[$i] == $ch) {
            $rs .= $str[$c];
            $c++;
        } else {
            $rs .= $mask[$i];
        }
    }

    return $rs;
}

$str = '85994407071'; // Exemplo para telefone
echo format_string('(##)-#####-####', $str);

$str = '12042024'; // Exemplo para datas
echo '<br />' . format_string('##/##/####', $str);
```

4 – Usando em PHP

Funções PCRE Objetivo

preg_filter Busca e substitui, retornando as opções do array que casarem com a expressão.
preg_grep Retorna as opções de um array que casarem com a expressão.
preg_last_error Retorna o código de erro da última expressão executada.
preg_match_all Retorna as ocorrências de uma string que casarem com a expressão.
preg_match Verifica se uma string casa com a expressão.
preg_quote Adiciona escape em caracteres da expressão.
preg_replace_callback Busca e executa um callback nas opções que casarem com a expressão.
preg_replace Busca e substitui, retornando todas as opções.
preg_split Divide uma string utilizando uma expressão.

Funções iniciadas com ereg_ estão defasadas

preg_match - Execute uma correspondência de expressão regular

Início de string ^
preg_match('/^rato/', $string)

Final de string $
preg_match('/rato$/', $string

Uma expressão OU outra: |
preg_match('/rato|gato/')

```
<?php
preg_match('/(foo)(bar)(baz)/', 'foobarbaz', $matches, PREG_OFFSET_CAPTURE);
print_r($matches);
```

Case sensitividade

```
<?php
// O "i" após o delimitador padão indica uma busca case-insensitiva

if (preg_match("/php/i", "PHP is the web scripting language of choice.")) {
 echo "A match was found.";
} else {
 echo "A match was not found.";
}
```

Algumas funções com regex:

Uma solução melhor para validar a sintaxe do email é usar filter_var.

```
if (filter_var('test+email@fexample.com', FILTER_VALIDATE_EMAIL)) {
- echo "Your email is ok.";
} else {
- echo "Wrong email address format.";
}
```

Valida o nome de usuário, consiste em caracteres alfanuméricos (a-z, A-Z, 0-9), sublinhados que tem no mínimo 5 caracteres e no máximo 20 caracteres.
Você pode alterar o caractere mínimo e o caractere máximo para qualquer número que desejar.

```
$username = "user_name12";
if (preg_match('/^[a-z\d_]{5,20}$/i', $username)) {
- echo "Your username is ok.";
} else {
- echo "Wrong username format.";
}
```

Validar domínio

```
$url = "http://komunitasweb.com/";
if (preg_match('/^(http|https|ftp):\/\/([A-Z0-9][A-Z0-9_-]*(?:\.[A-Z0-9][A-Z0-9_-]*)+):?(\d+)?\/?/i',
$url)) {
- echo "Your url is ok.";
} else {
- echo "Wrong url.";
}
```

Extraia o nome de domínio de determinado URL

```
$url = "http://komunitasweb.com/index.html";
preg_match('@^(?:http://)?([^/]+)@i', $url, $matches);
$host = $matches[1];
echo $host;
```

Destaque uma palavra no conteúdo

```
$text = "Sample sentence from KomunitasWeb, regex has become popular in web programming.
Now we learn regex. According to wikipedia, Regular expressions (abbreviated as regex or regexp,
with plural forms regexes, regexps, or regexen) are written in a formal language that can be
interpreted by a regular expression processor";
$text = preg_replace("/\b(regex)\b/i", '<span style="background:#5fc9f6">\1</span>', $text);
echo $text;
```

Obtenha todas as imagens de um documento HTML
Se você sempre quis obter todas as imagens de uma página da web, este código é essencial para você. Você deve criar facilmente um downloader de imagens usando o poder do cURL.

```
$images = array();
preg_match_all('/(img|src)=("|')[^"'>]+/i', $data, $media);
unset($data);
$data=preg_replace('/(img|src)("|'|="|=')(.*)/i',"$3",$media[0]);
foreach($data as $url)
{
   $info = pathinfo($url);
   if (isset($info['extension']))
   {
      if (($info['extension'] == 'jpg') ||
      ($info['extension'] == 'jpeg') ||
      ($info['extension'] == 'gif') ||
      ($info['extension'] == 'png'))
      array_push($images, $url);
   }
}
```

Corresponder valores de cores hexadecimais
Outra ferramenta interessante para desenvolvedores web! Ele permite combinar/validar um valor de cor hexadecimal.

```
$string = "#555555";
if (preg_match('/^#(?:(?:[a-fd]{3}){1,2})$/i', $string)) {
echo "example 6 successful.";
}
```

Verifique a complexidade da senha
Esta expressão regular testará se a entrada consiste em 6 ou mais letras, dígitos, sublinhados e hifens.
A entrada deve conter pelo menos uma letra maiúscula, uma letra minúscula e um dígito.

```
'A(?=[-_a-zA-Z0-9]*?[A-Z])(?=[-_a-zA-Z0-9]*?[a-z])(?=[-_a-zA-Z0-9]*?[0-9])[-_a-zA-Z0-9]
{6,}z'
```

Gere Emoticons Automaticamente
Outra função usada pelo WordPress. Este permite substituir automaticamente um símbolo de emoticon por uma imagem.

```
$texte='A text with a smiley :-)';
echo str_replace(':-)','<img src="smileys/souriant.png">',$texte);
$words = [ "Seven", "even", "Maven", "Amen", "Leven" ];
$pattern = "/.even/";
```

```
foreach ($words as $word) {

  if (preg_match($pattern, $word)) {
    echo "$word matches the pattern\n";
  } else {
    echo "$word does not match the pattern\n";
  }
}


$words = [ "color", "colour", "comic", "colourful", "colored", - "cosmos", "coloseum", "coloured",
"colourful" ];
$pattern = "/colou?r/";

foreach ($words as $word) {
  if (preg_match($pattern, $word)) {
    echo "$word matches the pattern\n";
  } else {
    echo "$word does not match the pattern\n";
  }
}
```

Temos quatro nove no array $words.

```
$pattern = "/colou?r/";


$names = [ "Jane", "Thomas", "Robert", "Lucy", "Beky",
- "John", "Peter", "Andy" ];

$pattern = "/Jane|Beky|Robert/";

foreach ($names as $name) {

  if (preg_match($pattern, $name)) {
    echo "$name is my friend\n";
  } else {
    echo "$name is not my friend\n";
  }
}
```

Temos oito nomes no array $names.

```
$pattern = "/Jane|Beky|Robert/";

$words = [ "sit", "MIT", "fit", "fat", "lot" ];

$pattern = "/[fs]it/";

foreach ($words as $word) {
```

```
  if (preg_match($pattern, $word)) {
    echo "$word matches the pattern\n";
  } else {
    echo "$word does not match the pattern\n";
  }
}
```

We define a character set with two characters.

```
$pattern = "/[fs]it/";


$emails = [ "luke@gmail.com", "andy@yahoocom", "34234sdfa#2345", "f344@gmail.com"];

# regular expression for emails
$pattern = "/^[a-zA-Z0-9._-]+@[a-zA-Z0-9-]+\.[a-zA-Z.]{2,18}$/";

foreach ($emails as $email) {

  if (preg_match($pattern, $email)) {
    echo "$email matches \n";
  } else {
    echo "$email does not match\n";
  }
}
```

Note that this example provides only one solution. It does not have to be the best one.

```
$pattern = "/^[a-zA-Z0-9._-]+@[a-zA-Z0-9-]+\.[a-zA-Z.]{2,18}$/";


$pattern = "/ca[kf]e/";
$text = "He was eating cake in the cafe.";
$matches = preg_match_all($pattern, $text, $array);
echo $matches . " matches were found.";


$pattern = "/color/i";
$text = "Color red is more visible than color blue in daylight.";
$matches = preg_match_all($pattern, $text, $array);
echo $matches . " matches were found.";
```

Alguns exemplos usando PHP

```
preg_match('/(foo)(bar)(baz)/', 'foobarbaz', $matches, PREG_OFFSET_CAPTURE);
//var_dump($matches);exit;
print $matches[0][0].'<br>';
print $matches[1][0].'<br>';
print $matches[2][0].'<br>';
print $matches[3][0].'<br>';
```

```
$subject = "abcdef";
$pattern = '/^def/';
preg_match($pattern, substr($subject,3), $matches, PREG_OFFSET_CAPTURE);
var_dump($matches);

$string = 'orato';
if (preg_match('/^rato/', $string)) {
- print 'A string comeca com "rato"';
} else {
- print 'A string nao comeca com "rato"';
}

$string = 'barato';
if (preg_match('/rato$/', $string)) {
- print 'A string termina com "rato"';
} else {
- print 'A string nao termina com "rato"';
}

$string = 'gato';
if (preg_match('/rato|gato/', $string)) {
- print 'A string contém gato';
} else {
- print 'A string não contém gato';
}

$string = 'gato';
// Exemplo: checar se a string comeca com "r" ou "g", seguido de "at" e termina com "a" ou "o". Somente at é fixo
if (preg_match('/^[rg]at[ao]$/', $string)) {
- print 'Sim';
} else {
- print 'Não';
}

// Exemplo: checar se a string comeca com "x", seguido por uma letra minuscula, depois um numero, e termina com
// uma letra ou numero ou @
if (preg_match('/^[x][[:lower:]][\d][a-z0-9@]$/', $string)) {
- // Por exemplo: "xa0a", "xb11", "xb1@"
}

// Mesmo exemplo, mas usando apenas sequencias de escape
if (preg_match('/^x\w\d[\w\d@]$/', $string)) {
- // Por exemplo: "xa0a", "xb11", "xb1@"
}

// Exemplo: a string precisa começar com "a" ou "b", depois seguir com qualquer caractere, exceto o "x" e "y"

if (preg_match('/^[ab][^xy]$/', $string)) {
 // Por exemplo: "a1", "bd"
```

```
} else {
  // Por exemplo: "ax", "ay", "bx", "by", "k1"
}

// Capturar o prefixo de uma placa de carro (3 letras) e o sufixo (4 numeros)
$string = 'ABC-1234';
if (preg_match('/^([A-Z][A-Z][A-Z])-([0-9][0-9][0-9][0-9])$/', $string, $partes)) {
  echo $partes[0]; // ABC-1234
  echo $partes[1]; // ABC
  echo $partes[2]; // 1234
}

// Capturar o prefixo de uma placa de carro (3 letras)
$string = 'ABC-1234';
if (preg_match('/^([A-Z][A-Z][A-Z])-(?:[0-9][0-9][0-9][0-9])$/', $string, $partes)) {
  echo $partes[0];     // ABC-1234
  echo $partes[1];     // ABC
}

// Capturar a sigla de segunda-feira ou terça feira
$string = 'terça-feira';
if (preg_match('/^(?|(seg)unda|(ter)ça)-feira$/', $string, $partes)) {
  echo $partes[0];     // ABC-1234
  echo $partes[1];     // ter
}

// Capturar o prefixo de uma placa de carro (3 letras) e o sufixo (4 numeros)
$string = 'ABC-1234';
if (preg_match('/^([A-Z]{3})-([0-9]{4})$/', $string, $partes)) {
  echo $partes[0]; // ABC-1234
  echo $partes[1]; // ABC
  echo $partes[2]; // 1234
}

// Checar se a string eh um CPF formatado
if (preg_match('/^\d{3}\.\d{3}\.\d{3}-\d{2}$/', $string, $partes)) {
  // O CPF esta formatado
}

$cpf = '12106836368';
$cpf = preg_replace("/[^0-9]/", "", $cpf);
echo $cpf;
```

para validar o nome do usuário quando é enviado a seu aplicativo:
^[A-Za-z][A-Za-z0-9_]{2,9}$.

```
// Checar se a string eh formada por 3 ou mais digitos, seguido ou nao de hifen,
// seguido de um ou mais digitos, seguido de 0 ou mais letras
if (preg_match('/^\d{3,}-?\d+[a-z]*$/', $string, $partes)) {
```

```
// Por exemplo: "1234", "123-4", "12345", "1234-567", "1234ab", "123-4a"
} else {
// Por exemplo: "12", "1-2", "12-3", "123-", "123-a"
}
```

```
<?php
// create a string
$string = 'abcdefghijklmnopqrstuvwxyz0123456789';

// try to match the beginning of the string
if(preg_match("/^abc/", $string))
  {
  // if it matches we echo this line
  echo 'The string begins with abc';
  }
else
  {
  // if no match is found echo this line
  echo 'No match found';
  }
?>
```

```
<?php
// create a string
$string = 'abcdefghijklmnopqrstuvwxyz0123456789';

// try to match our pattern at the end of the string
if(preg_match("/89\z/i", $string)) // z - finaliza
    {
  // if our pattern matches we echo this
    echo 'The string ends with 89';
    }else{
   // if no match is found we echo this line
    echo 'No match found';
    }
?>
```

```
<?php
// create a string
$string = 'big';

// Search for a match
echo preg_match("/b[aoiu]g/", $string, $matches);
```

```
<?php
// create a string
$string = 'abcefghijklmnopqrstuvwxyz0123456789';

// echo our string
preg_match("/[^b]/", $string, $matches);

// loop through the matches with foreach
foreach($matches as $key=>$value)
   {
   echo $key.' -> '.$value; // 0 -> a
   }
?>


<?php
// create a string
$string = 'abcefghijklmnopqrstuvwxyz0123456789';

// try to match all characters not within out pattern
preg_match_all("/[^b]/", $string, $matches);

// loop through the matches with foreach
foreach($matches[0] as $value)
   {
   echo $value;
   }
?>


<?php

// create a string
$string = '12345678';

// look for a match
echo preg_match("/1234-?5678/", $string, $matches);


<?php
// create a string
$string = 'abcdefghijklmnopqrstuvwxyz0123456789';

// try to match our pattern
if(preg_match("/^ABC/i", $string))
{
// echo this is it matches
echo 'The string begins with abc';
}
```

```
else
{
// if not match is found echo this line
echo 'No match found';
}


<?php
// create a string
$string = 'We will replace the word foo';

// substitute the word for and put in bar
$string = preg_replace("/foo/", 'bar', $string);

// echo the new string
echo $string;


<?php

// the string to match against
$string = 'The cat sat on the mat';

// match the beginning of the string
echo preg_match("/^The/", $string); // returns 1

// match the end of the string
echo preg_match("/mat\z/", $string); // returns 1

// match anywhere in the string
echo preg_match("/dog/", $string); // returns 0 as no match was found for dog.


<?php
$cpf = '12106836368';
$cpf = preg_replace("/[^0-9]/", "", $cpf);
echo $cpf;
```

para validar o nome do usuário quando é enviado a seu aplicativo:
^[A-Za-z][A-Za-z0-9_]{2,9}$.

```
<?php
$cep = '22710-045';
$names = array('Diogo', 'Renato', 'Gomes', 'Thiago', 'Leonardo');
$text = 'Lorem ipsum dolor sit amet, consectetuer adipiscing.';

// Validação de CEP
$er = '/^(\d){5}-(\d){3}$/';
if(preg_match($er, $cep)) {
```

```
- echo "O cep casou com a expressão.";
}
// Resultado: O cep casou com a expressão.

// Busca e substitui nomes que tenham "go", case-insensitive
$er = '/go/i';
$pregReplace = preg_replace($er, 'GO', $names);
print_r($pregReplace);
// Resultado: DioGO, Renato, GOmes, ThiaGO, Leonardo

// Busca e substitui nomes que terminam com "go"
$er = '/go$/';
$pregFilter = preg_filter($er, 'GO', $names);
print_r($pregFilter);
// Resultado: DioGO, ThiaGO

// Resgatar nomes que começam com "go", case-insensitive
$er = '/^go/i';
$pregGrep = preg_grep($er, $names);
print_r($pregGrep);
// Resultado: Gomes

// Divide o texto por pontos e espaços, que podem ser seguidos por espaços
$er = '/[[:punct:]\s]\s*/';
$pregSplit = preg_split($er, $text);
print_r($pregSplit);
// Resultado: Array de palavras

// callback, retorna em letras maiúsculas
$callback = function($matches) {
- return strtoupper($matches[0]);
};

// Busca e substitui de acordo com o callback
$er = '/(.*)go$/';
$pregCallback = preg_replace_callback($er, $callback, $names);
print_r($pregCallback);
// Resultado: DIOGO, Renato, Gomes, THIAGO, Leonardo
```

Checar se uma string segue determinado padrão

```
// Exemplo
$string = 'ABC-1234';

// Checar se a string segue o padrao de uma placa de carro
if (preg_match('/^[A-Z]{3}\-[0-9]{4}$/', $string)) {
- // A string eh uma placa de carro valida
} else {
- // A string nao eh uma placa de carro valida
}
```

Capturar pedaços de uma string

```
// Exemplo
$cpf = '123.456.789-01';

// Capturar o digito verificador do CPF
if (preg_match('/^[0-9]{3}\.[0-9]{3}\.[0-9]{3}\-([0-9]{2})$/', $cpf, $partes)) {
- $digito_verificador = $partes[1];
}
```

Substituir pedaços de uma string por outra sequência de caracteres

```
// Exemplo 1: substituir vogais por "x":
$string = 'abcdefgh123';
$string2 = preg_replace('/[aeiou]/', 'x', $string);

// Exemplo 2: remover numeros de uma string
$string = 'abc123';
$string2 = preg_replace('/[0-9]/', '', $string);

// Exemplo 3: transformar um CPF com numeros em um CPF formatado
$string = '12345678901';
$string2 = preg_replace('/^([0-9]{3})([0-9]{3})([0-9]{3})([0-9]{2})$/', '$1.$2.$3-$4', $string);

// Exemplo 4: transformar uma data no formato YYYY-MM-DD em DD/MM/YYYY
$string = '2015-02-01';
$string2 = preg_replace('/^([0-9]{4})\-([0-9]{2})\-([0-9]{2})$/', '$3/$2/$1', $string);

// Exemplo 5: Converter as vogais de uma string para maiuscula
$string = 'abcdefgh123';
$string2 = preg_replace_callback(
- '/[aeiou]/',
- function($partes) {
    return strtoupper($partes[0]);
- },
- $string
);
```

Quebrar uma string em um array utilizando uma expressão regular

```
// Exemplo: quebrar a string quando encontrar ponto ou virgula
$string = 'abc, def. ghi';
$array = preg_split('/[,\.]/', $string);

// Capturar o prefixo de uma placa de carro (3 letras) e o sufixo (4 numeros)
$string = 'ABC-1234';
if (preg_match('/^([A-Z][A-Z][A-Z])-([0-9][0-9][0-9][0-9])$/', $string, $partes)) {
- echo $partes[0]; // ABC-1234
- echo $partes[1]; // ABC
- echo $partes[2]; // 1234
}

// Capturar o prefixo de uma placa de carro (3 letras)
$string = 'ABC-1234';
if (preg_match('/^([A-Z][A-Z][A-Z])-(?:[0-9][0-9][0-9][0-9])$/', $string, $partes)) {
- echo $partes[0];     // ABC-1234
- echo $partes[1];     // ABC
}


<?php
$meuPost = nl2br($_POST["msg"]);
$pattern = array(
        "/\[b](.*?)\[\/b]/",
        "/\[i](.*?)\[\/i]/",
        "/\[img](.*?)\[\/img]/"
); //sim! pode definir arrays de coisas pra achar e/ou substituir!
$replace = array(
        "<b>$1</b>",
        "<i>$1</i>",
        "<img src=\"$1\" />"
)
echo preg_replace($pattern, $replace, $meuPost);

<?php
// Um variável qualquer
$texto = 'Eu sou lindo!';

// Expressão regular
$expressao = '/lindo/';

// Retorna true se a expressão existir no texto
$verifica = preg_match( $expressao,  $texto);

// Verifica se existe
if ( $verifica ) {
```

```
	echo "A expressão $expressao existe no texto.";
} else {
	echo "A expressão $expressao não existe no texto.";
}

<?php
// Um texto que tem uma data
$texto = 'Eu nasci em 20/04/1987. Fiz 27 anos em 2014.';

// Expressao regular
$expressao = '/[0-9]{2}/[0-9]{2}/[0-9]{4}/';

// Verifica se existe
if ( preg_match( $expressao, $texto, $encontrado ) ) {
	// Retorna: A data 20/04/1987 foi encontrada no texto.
	echo "A data $encontrado[0] foi encontrada no texto.";
}

<?php
// Poema de Cecília Meireles - Prelúdio
$preludio = 'Que tempo seria,
ó sangue, ó flor
em que se amaria
de amor!

Pérolas de espuma,
de espuma e sal.
Nunca mais nenhuma
igual.';

// Expressao regular
$expressao = "/s/";

// Altera o texto original
$preludio = preg_replace(
	$expressao,
	'',
	$preludio
);

// Exibe o texto
echo $preludio;
```

```
<?php
$str = "Visit W3Schools";
//$pattern = "/w3schools/"; // 0
$pattern = "/w3schools/i"; // 1
echo preg_match($pattern, $str);
```

```
<?php
$pattern = "/exemplo/";
$subject = "Casa com a palavra exemplo";
$matches = array();

// Executa nossa expressão
$resultado = preg_match($pattern, $subject, $matches);

if ($resultado === 1) {
- print "casou<br>";
- var_dump($matches);

} else if ($resultado === 0) {
- print "não casou";
- var_dump($matches);

} else if ($resultado === false) {
- print "ocorreu um erro";

}
```

```
<?php
$regExp = "/[a-zA-Z]+ \d+/";
if (preg_match($regExp, "Januaury 26")) // This statement matches the regular expression with the string, If mathces then if statement executes, otherwise else statment.
{
  echo "The regular expression permit string.";
} else {
  echo "The regex pattern does not match with the string.";
}
```

Aqui para telefone de qualquer tipo:

```
function validaTelefone($t){
     return (bool) preg_match('/[0-9]{10,11}/', preg_replace('/\D/', '', $t));
}
```

Aqui apenas para celulares quem tiverem o 9º (nono) dígito preenchido:

```
function validaCelular($c){
     return (bool) preg_match('/[0-9]{2}[9][6789][0-9]{3}[0-9]{4}/',
preg_replace('/\D/', '', $c));
}
```

5 – Usando em Javascript

Como Funcionam as Expressões Regulares em JavaScript

No JavaScript, as expressões regulares são integradas na linguagem, permitindo uma ampla gama de operações de busca e manipulação de strings.

Sintaxe Básica

A sintaxe básica de uma expressão regular em JavaScript é /padrão/modificadores. Por exemplo, /abc/g procura todas as ocorrências da string "abc".

Validando Emails

Uma expressão regular (regex) para validar endereços de e-mail é um padrão que busca corresponder à estrutura típica de um e-mail. Uma expressão regular simples, mas relativamente eficaz para validar e-mails poderia ser:

^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$

Vamos quebrá-la em partes:

- ^[a-zA-Z0-9._%+-]+: Esta parte da regex corresponde ao início do e-mail. Ela valida que o e-mail deve começar (^) com uma ou mais (+) ocorrências de caracteres alfanuméricos (a-zA-Z0-9), pontos (.), sublinhados (_), percentuais (%), sinais de mais (+) ou hifens (``).

- @: Este caractere é literal e separa o nome do usuário do domínio do e-mail.

- [a-zA-Z0-9.-]+: Após o @, esta parte corresponde ao domínio do e-mail. Ela permite uma ou mais ocorrências de caracteres alfanuméricos, pontos ou hifens.

- \\.[a-zA-Z]{2,}$: Finalmente, esta parte corresponde ao TLD (top-level domain) do e-mail. Ela espera um ponto (\\.) seguido por pelo menos dois caracteres alfabéticos ([a-zA-Z]{2,}). O $ no final da regex indica que o TLD deve ser no final do e-mail.

- Esta expressão regular abrange a maioria dos endereços de e-mail usuais, mas vale ressaltar que, devido à natureza complexa e variada dos endereços de e-mail, pode haver casos extremamente raros ou específicos que ela não consiga validar.

Extração de Números de Texto

Para extrair números de uma string:

const numRegex = \d{1,3}(?:[.,]\d{3})*(?:[.,]\d+)?

const nums = 'Exemplo 1234'.match(numRegex);

Dicas e Melhores Práticas

As expressões regulares são poderosas, mas devem ser usadas com cuidado para evitar complexidades desnecessárias e problemas de desempenho.

Evitar Erros Comuns

Teste suas expressões regulares extensivamente para garantir que elas funcionem como esperado em todos os casos de uso.

Otimizando o Desempenho

Considere a complexidade das suas regex. Expressões muito complexas podem ser lentas e difíceis de manter.

Ferramentas e Recursos Úteis

Use ferramentas online como Regex101 ou Regexr para construir e testar suas expressões regulares.

Desafios e Soluções Frequentes

Ao lidar com casos mais complexos, divida o problema e construa a regex passo a passo.

https://blog.rocketseat.com.br/expressoes-regulares-em-javascript/

Exemplos

```
<script>
var string = "Our Site is helpfull for studying about technical courses.!";
pattern="technical";
var res = string.search(pattern); /* This statement stores the position of the pattern in a string, if it is found in a string. */
document.write("Position of the pattern in a string:");
document.write(res);
</script>
```

```
<script>
var string = "You are a Bad Student";
var pattern=/Bad/;
var replace="Good";
var res = string.replace(/Bad/,replace);-
/* The above statement replaces the Bad word from the string by the Good word using the replace method. */
document.write("After replacing the substring, the modified string is:"+ '<br>');
document.write(res);
</script>
```

Extra

Máscara

```
<script>
function format(mask, number) {
   var s = ''+number, r = '';
   for (var im=0, is = 0; im<mask.length && is<s.length; im++) {
     r += mask.charAt(im)=='X' ? s.charAt(is++) : mask.charAt(im);
   }
   return r;
}

console.log(format('XXXX.XXXX.XXXX-X', 1234213412341)); // esse primeiro é o que
vc quer
console.log(format('XX/XX/XX/XX/XX/XX,X', 1234213412341));
console.log(format('XX muito XX manero XX.XX.XX.XX.X', 1234213412341));
</script>



function formatar_valor(str) {
    var split = str.match(/.{1,4}/g),          // divide a string em blocos de 4
        join  = split.join('.');               // junta os blocos colando-os com
um .

    return join.replace(/\.(?!.*?\.)/, '-');   // substitui o ultimo . por um -
}

alert(formatar_valor('1234213412341'));        // devolve: 1234.2134.1234-1
```

6 – Usando em MySQL/MariaDb

O MySQL suporta um tipo de operação de busca de padrões baseada em expressões regulares com o operador REGEXP.

Exemplos de consultas usando Expressões Regulares no MySQL:

```
create table livros(
id int primary key auto_increment,
titulo varchar(50) not null
);

insert into livros (titulo) values ('Meu pé de laranja lima');
insert into livros (titulo) values ('Quo vadis');
insert into livros (titulo) values ('A Bíblia');
insert into livros (titulo) values ('Gabriela, cravo e canela');
insert into livros (titulo) values ('Não lembro');
```

1. Retornar os nomes dos livros da tabela livros, onde o nome do livro se inicie com uma das letras F ou S

```
SELECT titulo
FROM livros
WHERE titulo REGEXP '^[FS]';
```

```
SELECT titulo
FROM livros
WHERE titulo REGEXP '^[QU]';
```

2. Trazer os livros cujos nomes não se iniciam nem com o caractere Q, nem com o caractere U

```
SELECT titulo
FROM livros
WHERE titulo REGEXP '^[^QU]';
```

3. Retornar titulos de livros que finalizem com as letras o ou s:

```
SELECT titulo
FROM livros
WHERE titulo REGEXP '[os]$';
```

4. Consultar os livros cujos nomes comecem com as letras G ou N, ou ainda com a sequência de caracteres Br

```
SELECT titulo
FROM livros
WHERE titulo REGEXP '^[GN]|Br';
```

7 – Outros usos

gedit

Ctrl+H - opção para localizar e sobrescrever. Tem opção de Expressões Regulares

vscode

Ctrl+H

Aparecem as caixas Find e a Replace. Na Find tem .* ao final para regex

LibreOffice Writer

Ctrl+H também tem Expressões regulares

Fabrik no Joomla

Máscaras de entrada de campos dos formulários

Um exemplo no e-book
https://github.com/ribafs2/fabrik/blob/main/fabrik4-ebook.pdf

Página 17

Vídeo
https://www.youtube.com/watch?v=-WT_GMgbWGI&t=282s

8 – Exercício Prático

Procurar e sobrescrever usando Expressões regulares

Se temos uma lista de palavras na vertical, assim:

palavra1
aplavra2
palavra3
etc

E queremos criar uma string na horizontal formada por cada palavra delimitada por aspas e separadas por vírgula, assim:

'palavra1', 'aplavra2', 'palavra3', ...

- Copiamos e colamos a lista de palavras em um novo arquivo do VSCode do tipo Plain text

- Teclamos Ctrl+H para procurar e sobrescrever

- Na caixa de cima entramos com \n para que ele busque pelo caractere de final de linha. Lembrar de clicar no asterismo ao final para que a busca use regex

- Na caixa de baixo entrar com ', (apóstrofo e vínrgula)

- Então clicar em Replace All. Ficará assim, na horizontal:

palavra1',aplavra2',palavra3',...

Agora precisaremos adicionar uma aspa ao início das palavras. Então tecle Ctrl+H novamente e na caixa superior, limpe e entre uma vírgula

- Na caixa de baixo entre , ' (vírgula, espaço e apóstrofo) e ficará assim:

palavra1', 'aplavra2', 'palavra3', ...

Para que fique completa resta apenas entrar manualmente com um apóstrofo (aspa simples) no inínio da primeira palavra e ao final da última.

Eliminar todas as linhas em branco de um arquivo ou de sequência de palavras

Se tenho a sequência de palavras:
palavra1
aplavra2
palavra3
palavra4
palavra5

Para remover as linhas em branco copie e cole num arquivo tipo Plain text do VSCode

Tecle Ctrl+H e na caixa superior digite

^(?:[\t]*(?:\r?\n|\r))+

Na de baixo deixe em branco

Então clique em Replace All

Remover todas as linhas em branco de um arquivo no VSCode

Ctrl+H

^(?:[\t]*(?:\r?\n|\r))+

9 – Testadores de regex

https://regex101.com/

https://www.regextester.com/

https://www.phpliveregex.com/

https://ribafs2.github.io/regex-tester/

https://regexr.com/

- Observe que, por padrão, os sinalizadores m e g estão habilitados em regex101.com. Portanto, se você usar ^ e $, eles corresponderão no início e no final das linhas de forma correspondente. Se você precisar do mesmo comportamento em seu código, verifique como o modo multilinha é implementado e use um sinalizador específico ou - se suportado - use um modificador embutido (?m) embutido (inline). O sinalizador g permite a correspondência de múltiplas ocorrências e geralmente é implementado usando funções/métodos específicos. Verifique a referência do seu idioma para encontrar o idioma apropriado.

https://www.regexpal.com/

Testador online
https://regex101.com/

Na caixa de texto acima digite:

/Bar8/

Na textarea abaixo ele somente reconhecerá quando você digitar
Bar8

Digite acima:

/[BLC]ar8/
```php
Abaixo:
Bar8
Lar8
Car8
LCar8
BCar8
BLCar8
```
Ou em outra ordem as 3 primeiras

Digite acima: ar são obrigatórias, BLC opcionais, apenas uma obrigatória. 854 opcionais, apenas uma obrigatória
/[BLC]ar[854]/
```php
Abaixo:
Bar8

Faixa A-Z

Digite acima
/[A-Z]ar8/

Abaixo uma:
Bar8

Acima:
/[A-Z-a-z]ar8/

Abaixo:
bar8

Acima
/[A-Z]ar[0-9]/

/[^CL]ar8/  (Não pode começar com C nem L)

/.ar8/  (O ponto diz que primeiro dígito pode letra, número ou símbolo, qualquer coisa)

/[A-Z][a-z][a-z][0-9]/  (Quatro dígitos)

/[A-Z][a-z]*[0-9]/

Mam7- (O terceiro dígito, *, pode ser qualquer dígito e este pode se repetir
Mamdmdmdmdmd4

/[0-9]{3}/-   (De 0 a 9 sendo 3 dígitos)
000
123

/[0-9]{2,3}/

121-068-363-11

/[0-9]{3}\.?[0-9]{3}\.?[0-9]{3}-?[0-9]{2}/

121.069.343-56

/[0-9]{3}\.?[0-9]{3}\.?[0-9]{3}-?([0-9]{2})/
```

/[0-9]{3}\.?[0-9]{3}\.?[0-9]{3}-?([0-9]{2})?/

String
\w

Bar8

Dígito
\d

10 – Boas Referências

https://www.oficinadanet.com.br/artigo/programacao/o-que-sao-expressoes-regulares

https://catswhocode.com/php-regex/

https://rubsphp.blogspot.com/2015/02/expressoes-regulares-em-php.html

https://bar8.com.br/regex-expressoes-regulares-sap-abap-cdcb3271dd67

https://www.littlebigtomatoes.com/easily-remove-empty-lines-using-regular-expression-in-visual-studio-code/

https://www.youtube.com/watch?v=_YUqYkm8u9o

https://www.bosontreinamentos.com.br/mysql/mysql-regexp-expressoes-regulares-em-consultas-23/

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions

https://builtin.com/software-engineering-perspectives/javascript-regex

https://www.freecodecamp.org/news/regular-expressions-for-javascript-developers/

https://eloquentjavascript.net/09_regexp.html