TECNOLOGIA EM SISTEMAS PARA INTERNET
ESCOLA DO MAR, CIÊNCIA E TECNOLOGIA
UNIVALI

PROGRAMAÇÃO WEB

Professor: Carlos Henrique Bughi

Expressões regulares

Introdução

- O que é?

  Uma expressão regular, também conhecida como “regex”, é um padrão que caracteriza uma certa porção de texto.

Exemplo:

A expressão regular “ato”, pode representar um pedaço das seguintes palavras:

“gato”,“prato”, “ator”, “infratores”, etc...

Introdução

- Aonde utilizar expressões regulares?

  Em linguagens com suporte à expressões regulares como por exemplo:

  PHP, Java, Perl, Python, .NET, ruby, javascript.

Para que servem as regex?

- Servem para procurar e comparar caracteres que seguem um determinado padrão (pattern) em uma string.

Exemplos:

Verificar se uma string começa com uma determinada letra.

Verificar se uma string possui apenas caracteres numéricos.

Verificar se uma string termina com um ponto final.

Verificar se uma string possui uma letra seguida de um número.

E muitas outras coisas...

Vantagens da regex

- Código se torna mais enxuto.
- Código se torna mais legível.
- A manutenção do código se torna mais fácil.

Sintaxe da regex

• Alguns detalhes podem variar de acordo com a linguagem utilizada, mas a essência é a mesma.

Nesta apresentação será usado como exemplo a sintaxe do javascript.

então:

```
var minhaRegex = /^oi$/;
minhaRegex.test(“texto para analise”)
```

Sintaxe da regex

- Basicamente, um regex é formado por:
- Âncoras – onde?
- Representantes – quais caracteres?
- Quantificadores – quantos caracteres?

Âncoras

- Uma âncora indica que uma seqüência de caracteres deve estar no início ou no final de uma string.
- Exemplos:

  /^A/ : representa um “A” maiúsculo no começo de uma string.

  /ado$/ : representa a seqüencia de caracteres “ado” que está no final de uma string.

símbolo	significado
^	Os caracteres à direita deste símbolo, devem estar no começo da string.
$	Os caracteres à esquerda deste símbolo, devem estar no final da string.

Representantes

Um representante é um elemento que representa um conjunto de caracteres.

•O representante “.” (necessitado) :

O caractere “.” (ponto final), dentro de uma expressão regular, representa qualquer caractere não importando se o caractere é uma letra, número ou qualquer outra coisa.

Exemplos: /^mat.$/ -> representa as strings:

“mata”, “mate”, “mato”, “matO”, “matx”, “mat9”, “mat “(note o espaço no final).

/^mat.$/ -> **NÃO** representa as strings:

“matar”, “matei”, “mat” (note que esta string não possui espaço no final).

Representantes

- O representante “[]” (exigente) :

  Usado para representar um grupo de caracteres a ser procurado no texto. Cuidado com o "-" dentro desse grupo, pois ele faz com que a lista fique maior, [a-z] procurando por letras de "a" a "z".

  Exemplos: /^[bcfgmr]ala$/ -> representa as strings:

  “bala”, “cala”, “fala”, “gala”, “mala” e “rala”

  /^usuario[0-8]$/ -> representa as strings:

  “usuario0”, “usuario1”, “usuario2”, “usuario3”, ..., “usuario8”

Representantes

- O representante “[^]” (lista negada) :

  Usada para representar todos os caracteres que **NÃO** fazem parte da lista de caracteres indicada.

  Exemplo: /^[1-5][^0]$/ -> representa as strings:

  “11”, “2!”, “3g”, “53”...

  Ou seja, representa qualquer string que começa com um número de 1 a 5 seguido de qualquer caractere diferente de 0.

Quantificadores

- Servem para controlar a quantidade de ocorrências de um caractere ou de um conjunto de caracteres.

- Quantificador “?” (opcional):

  O caractere ou grupo de caracteres que precede este quantificador torna-se opcional.

  Exemplos:

  /^balas?$/ -> representa as strings:

  “bala” e “balas”.

  Ou seja, representa a palavra “bala”, seguida ou não de “s”.

  /^usuario[1-3] ?$/ -> representa as strings:

  “usuario1”, “usuario2”, “usuario3” e “usuario”.

  Ou seja, representa a palavra “usuario”, seguida ou não de um número de 1 a 3.

Quantificadores

- Quantificador “*” (tanto faz):

  O caractere ou grupo de caracteres que precede este quantificador torna-se opcional, e caso ele exista, pode ser repetido qualquer número de vezes.

  Exemplo:

  /^beijos*$/ -> representa as strings:

  “beijo”, “beijos”, “beijosssss”, “beijosssssssssssss”

  /^usuario[a-d1-4]*$/ -> representa as strings:

  “usuario”, “usuarioccc”, “usuario33333”, etc...

  mas NÃO representa as strings:

  “usuarioabc”, “usuariod3”, etc...

Quantificadores

- Quantificador “+” (tem-que-ter):

  O caractere ou grupo de caracteres que precede este quantificador torna-se **obrigatório**, e caso ele exista, pode ser repetido qualquer número de vezes.

  Exemplo:

  /^beijos+$/ -> representa as strings:

  “beijos”, “beijosssss”, “beijosssssssssssss”

  mas NÃO representa a string:

  “beijo”.

Quantificadores

- Quantificador “{n,m}”, “{n}”, {n,}, ou {,m} (controle):

  O caractere ou grupo de caracteres que precede este quantificador deve se repetir um número x de vezes, tal que x >= n e x <= m.

  Se **n** for omitido, então não haverá um valor mínimo de repetições.

  Se for utilizado {**n** }, então a repetição deverá ocorrer exatamente **n** vezes.

  Exemplo:

  /^beijos{2,4}$/ -> representa as strings:

  “beijoss”, “beijosss” e “beijossss”.

  mas NÃO representa a string:

  “beijos” nem “beijosssssssssss”.

Modificadores

- Utilizados para realizar consultas case-sensitive ou globais.

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

```
var patt = /padrao/i
```

Escapando caracteres reservados

- Alguns caracteres como:

  “.”, “[”, “/”, “\”, “(”, “$” são reservados para o uso de quantificadores, representantes, etc...

  Para capturar estes caracteres especiais usando-se uma regex, basta adicionar o caractere de escape “\” antes do caractere especial.

  Exemplo:

  A expressão regular /^\.oi\\$/ irá representar a string:

  “.oi\”

Mais regex javascript

- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Regular_Expressions

Metodo	Descrição
exec	Um método RegExp que execute uma pesquisa por uma correspondência em uma string. Retorna um array de informações.
test	Um método RegExp que testa uma correspondência em uma string. Retorna true ou false.
match	Um método String que executa uma pesquisa por uma correspondência em uma string. Retorna uma array de informações ou null caso não haja uma correspondência.
search	Um método String que testa uma correspondência em uma string. Retorna o indice da correspondência ou -1 se o teste falhar.
replace	Um método String que executa uma pesquisa por uma correspondência em uma string, e substitui a substring correspondênte por uma substring de substituição.
split	Um método String que usa uma expressão regular ou uma string fixa para quebrar uma string dentro de um array de substrings.

Atividade

- Editar o arquivo atividade.html para que as expressões regulares atendam os requisitos comentados no código fonte.
- **Observação: Alterar apenas as linhas que possuem o comentário “//alterar esta linha”**

Referências

- http://www.numaboa.com/content/view/814/252/
- http://tools.lymas.com.br/regexp_br.php