

# Sistemas para Internet I

## Expressões Regulares

Profa. Diana F. Adamatti

Prédio do Centro de Ciências Computacionais –  
C3 1º. Andar – Sala 121

dianaadamatti@furg.br dianaada@gmail.com

Expressões Regulares

Exercícios

# Expressões Regulares

---

- Expressões regulares são formas concisas de descrever um conjunto de strings que satisfazem um determinado padrão.
- Por exemplo:
  - Podemos criar uma expressão regular para descrever todas as strings que representam datas no formato `dd/dd/yyyy`, onde `d` é um dígito qualquer.
  - Podemos verificar se uma string contém um número de telefone, descrito por uma expressão regular.

- Note que números de telefones e datas podem ser escritos em vários formatos diferentes.
- Números de telefones:
  - 19-91234-5678
  - (019) 91234 5678
  - (19)912345678
- Datas:
  - 09/10/2019
  - 09-10-19
  - 2019-10-09

- Expressões regulares constituem uma mini-linguagem, que permite especificar as regras de construção de um conjunto de strings.
- Essa mini-linguagem de especificação é muito parecida entre as diferentes linguagens de programação que possuem o conceito de expressões regulares (também chamado de RE, REGEX ou RegExp).
- Assim, aprender a escrever expressões regulares em JavaScript, será útil para descrever expressões regulares em outras linguagens de programação.
- Expressões regulares são frequentemente utilizadas para encontrar ou extrair informações de textos (*text parsing*).

- Exemplo de expressão regular:

```
1 '\d+\\\\'
```

- Essa expressão regular representa uma sequência de um ou mais dígitos seguidos por uma contrabarra (\).
- Vamos aprender regras de como escrever e usar expressões regulares.

- Letras e números em uma expressão regular representam a si próprios.
- Os caracteres especiais (chamados de meta-caracteres) são:

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

- . um caractere qualquer.
- ^ o início da string.
- \$ o fim da string.
- ? repetir zero ou uma vez.
- \* repetir zero ou mais vezes.
- + repetir uma ou mais vezes.
- \ usado para indicar caracteres especiais.

[ ] indica um conjunto de caracteres.

- `d[0-9]`: um dígito.
- `d[^0-9]`: um caractere que não é um dígito.
- `d[a-z]`: uma letra minúscula de a até z.
- `d[A-Z]`: uma letra maiúscula de A até Z.
- `d[a-zA-Z]*`: zero ou mais letras.
- `d[ACTG]+`: uma sequência de DNA.

{ } indica a quantidade de vezes que o padrão será repetido.

- `d[0-9]{2}`: dois dígitos.
- `d[a-z]{3}`: três letras minúsculas.
- `d[A-Z]{2,3}`: duas ou três letras maiúsculas.
- `d.{4,5}`: quatro ou cinco caracteres quaisquer.
- `d[01]{3,}`: pelo menos três bits.
- `d[0-9]{,6}`: no máximo seis dígitos.

( ) indica um grupo em uma expressão regular.

- `d([0-9]{3}\.){2}[0-9]{3}-[0-9]{2}`: um CPF.
- `d([a-z]+, )*[a-z]+`: uma sequência de uma ou mais palavras separadas por vírgulas (e espaços).

| similar ao operador lógico `or` para expressões regulares.

- `d([0-9]{3}|[a-z]{4})'`: uma sequência de três dígitos ou uma sequência de quatro letras minúsculas.

- Classes pré-definidas de caracteres:

`\d` um dígito, ou seja, `[0-9]`.

`\D` o complemento de `\d`, ou seja, `[^0-9]`.

`\s` um espaço em branco, ou seja, a `[\t\n\r\f\v]`.

`\S` o complemento de `\s`, ou seja, `[^\t\n\r\f\v]`.

`\w` um caractere alfanumérico, ou seja, `[a-zA-Z0-9_]`.

`\W` o complemento de `\w`, ou seja, `[^a-zA-Z0-9_]`.

## Exercícios

---

1. Escreva uma função para determinar se uma string representa um número (inteiro ou real) válido. Exemplos de números válidos: 10, +5, -3, -10.3, 0.80, 2.8033.
2. Escreva uma função para determinar se uma string representa um número de telefone (fixo ou celular) válido. Exemplos de números de telefones válidos:
  - (53) 3123-4567
  - 533123-4567
  - (053)3123-4567
  - (53)31234567
  - 5331234567
  - (053) 91234 5678
  - (053)91234 5678
  - 053912345678
  - 53 91234 5678
  - 5391234 5678

- `regexp = d^[+-]?[0-9]+(\.[0-9]+)?$`
- `ddd = d^(0?[1-9]{2}[- ]?|\(0?[0-9]{2}\) ?)`
- `tel = d[2-9]?[0-9]{4}[- ]?[0-9]{4}$`
- `regexp = ddd + tel`