

Expressões Regulares



- Aula 05 -
Coleta, Preparação e
Análise de Dados

Prof. Me. Lucas R. C. Pessutto



PUCRS

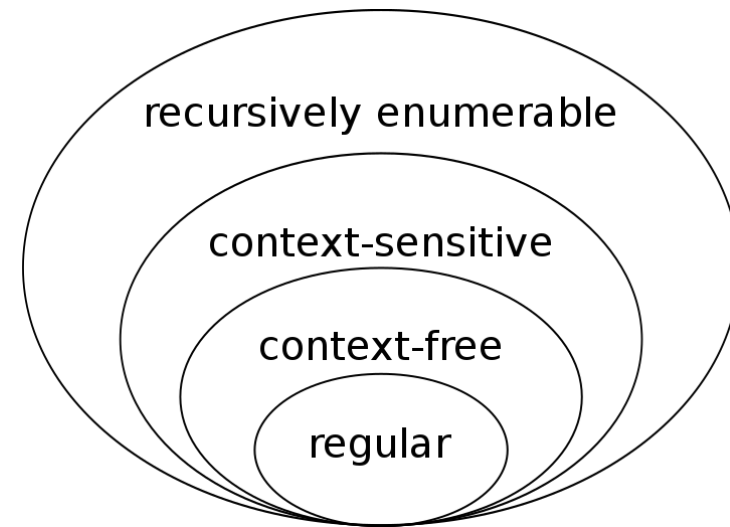
Pontifícia Universidade Católica
do Rio Grande do Sul



Slides adaptados do material do Prof. Lucas Silveira
Kupssinskü e do Prof. Luan Fonseca Garcia

Expressões Regulares

- Também conhecidas como regex
- É uma forma eficiente e concisa de fazer *matching* de strings
- Define um **padrão** que serve para filtrar texto.
- É uma forma eficiente e concisa de fazer *matching* de strings;
- Se os dados correspondem ao padrão da expressão regular, então ele é aceito para processamento.
- É um recurso poderoso mas um pouco “cryptic”



regex – lugares para testar

- <https://regex101.com/>
- vscode
- grep (com a opção -P)

Expressões Regulares

- Expressões Regulares fazem uso de *caracteres wildcard* (ou caracteres coringa, ou metacaracteres) para representar um ou mais caracteres na stream de dados.

REGULAR EXPRESSION

```
:r" teste
```

TEST STRING

```
teste•neste
```

REGULAR EXPRESSION

```
:r" neste
```

TEST STRING

```
teste•neste
```

REGULAR EXPRESSION

```
:r" [tn]este
```

TEST STRING

```
teste•neste
```

- Com apenas um comando e uma ER ***[nt]este*** conseguimos reconhecer o valores **neste** e **teste**.

Padrão de Texto Simples

- O padrão de regex mais simples é uma sequência de texto.
- A engine apenas verifica se os dados avaliados contém aquele pedaço de texto.
- ER são *case sensitive*
- O match entre o texto e a regex também pode ser parcial, como no caso de “test”.

REGULAR EXPRESSION

```
:r" Teste
```

TEST STRING

```
Texto•simples•para•um•teste•simples
```

REGULAR EXPRESSION

```
:r" teste
```

TEST STRING

```
Texto•simples•para•um•teste•simples|
```

REGULAR EXPRESSION

```
:r" test
```

TEST STRING

```
Texto•simples•para•um•teste•simples|
```

Caracteres Especiais

- Expressões regulares reservam a alguns caracteres um significado especial, diferente do seu significado original.

- São eles:

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

- Para manter o significado original de qualquer um desses caracteres precisamos utilizar o caractere de escape \

```
REGULAR EXPRESSION pattern error  
:r" [  
TEST STRING  
0•conjunto•tinha•valores•[0,10]
```

```
REGULAR EXPRESSION 1 match  
:r" \[  
TEST STRING  
0•conjunto•tinha•valores•[0,10]
```

Exercício

REGULAR EXPRESSION

TEST STRING

```
Qual•o•sentido•da•vida?↵  
Quem•eu?•Quem•mais?
```


Caracteres Âncora

- Quando definimos uma ER ela pode acontecer em qualquer posição do texto.
- Para especificarmos uma posição utilizamos os caracteres âncora.
- O caractere âncora para a posição inicial no texto é “^”.

REGULAR EXPRESSION	REGULAR EXPRESSION	REGULAR EXPRESSION	REGULAR EXPRESSION
<code>:r" testar</code>	<code>:r" ^testar</code>	<code>:r" Vamos</code>	<code>:r" ^Vamos</code>
TEST STRING	TEST STRING	TEST STRING	TEST STRING
<code>Vamos • testar • os • caracteres • • ancora. </code>	<code>Vamos • testar • os • caracteres • • ancora. </code>	<code>Vamos • testar • os • caracteres • • ancora. </code>	<code>Vamos • testar • os • caracteres • • ancora. </code>

Caracteres Âncora

- O caractere âncora para a última posição no texto é “\$”.

REGULAR EXPRESSION	REGULAR EXPRESSION	REGULAR EXPRESSION	REGULAR EXPRESSION
<code>:r" test\$</code>	<code>:r" ancora</code>	<code>:r" ancora\$</code>	<code>:r" ancora.\$</code>
TEST STRING	TEST STRING	TEST STRING	TEST STRING
Vamos•testar•os•caracteres •ancora.	Vamos•testar•os•caracteres •ancora.	Vamos•testar•os•caracteres •ancora.	Vamos•testar•os•caracteres •ancora.

- Também é possível combinar na mesma ER os caracteres de início e de fim.
- Esta é uma maneira efetiva de encontrar linhas em branco, por exemplo.
- Para isso, a ER seria: “^\$”

regex - âncoras

Usados para “marcar” uma parte da string.

Símbolo	Significado
<code>^</code>	Início da linha
<code>\$</code>	Fim da linha
<code>\b</code>	Início ou fim de palavra
<code>\B</code>	Não no começo nem no fim de palavra

Exemplo: `^teste$`

Exercício

REGULAR EXPRESSION

TEST STRING

texto•sem•contexto•texto•sem•pretexto↵

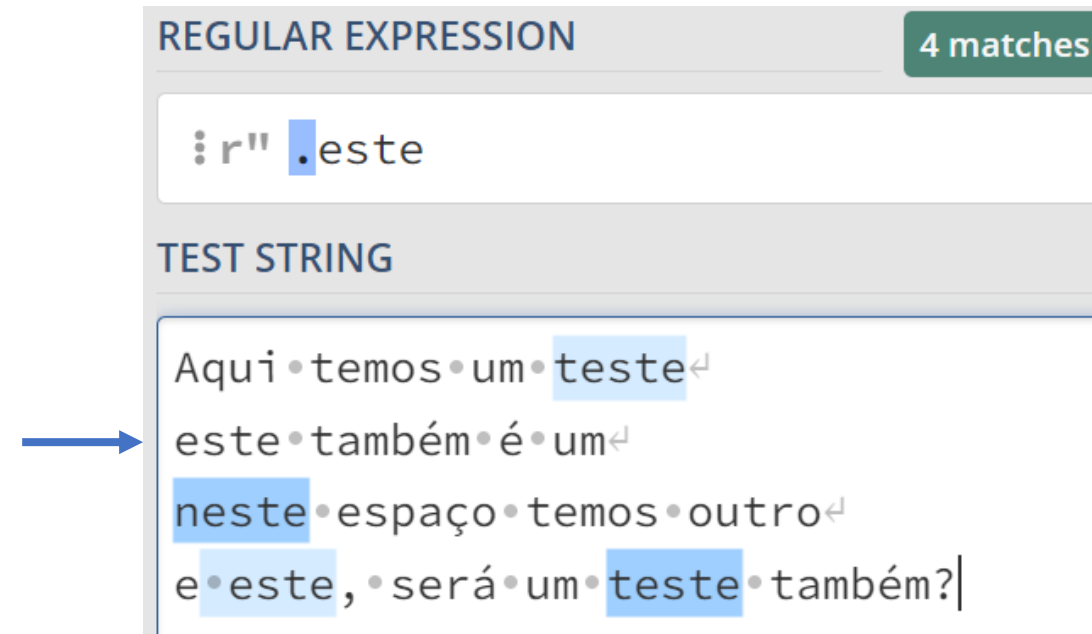
REGULAR EXPRESSION

TEST STRING

texto•sem•contexto•texto•sem•pretexto↵

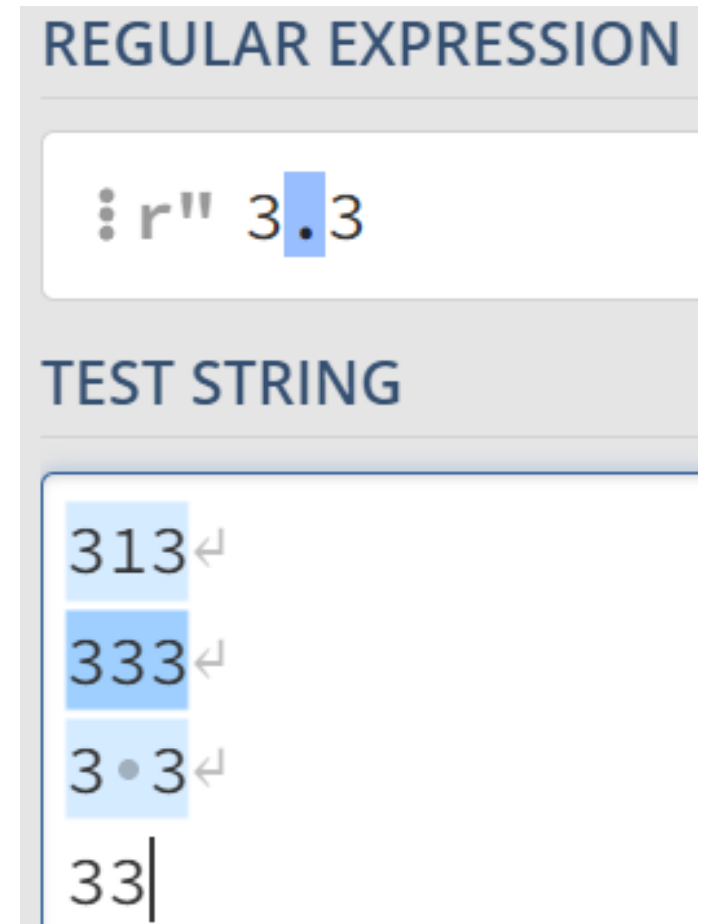
Caractere . (ponto)

- O . (ponto) pode ser utilizado para encontrar qualquer caractere único **exceto** o caractere de nova linha.
- A segunda linha não passa no teste da RE, pois não possui nenhum caractere antes de “este”.
- Já a quarta, passa, por quê?
- Porque o espaço em branco também é um caractere!



Caractere . (ponto)

- Também podemos procurar por caracteres entre outros dois caracteres.
- O único caso em que a ER não é satisfeita é o último, pois não existe nenhum caractere entre dois dígitos 3.



Exercício

REGULAR EXPRESSION

TEST STRING

19/08/2024 • ↵

9/6/2023 ↵

09/06/2023 ↵

14/04/18 ↵

14/04/2018

Classes de Caracteres

- O . é bastante útil, mas aceita qualquer caractere.
- E se quisermos limitar quais caracteres são aceitos?
- Para isto, podemos utilizar uma **classe de caracteres**.
- Colocamos todos caracteres dentro de colchetes “[]”, que serão tratados como qualquer outro caractere coringa.

Classes de Caracteres

- Por exemplo, a classe `[tn]` aceita textos em que a palavra “este” é precedida do caractere **t** ou do caractere **n**
- Podemos utilizar também para testar maiúsculas ou minúsculas:
- Também podemos combinar mais de uma classe:

REGULAR EXPRESSION

```
: r" [tn]este
```

TEST STRING

```
neste•teste•peste|
```

REGULAR EXPRESSION

```
: r" [Ss]im
```

TEST STRING

```
Sim•sim|
```

REGULAR EXPRESSION

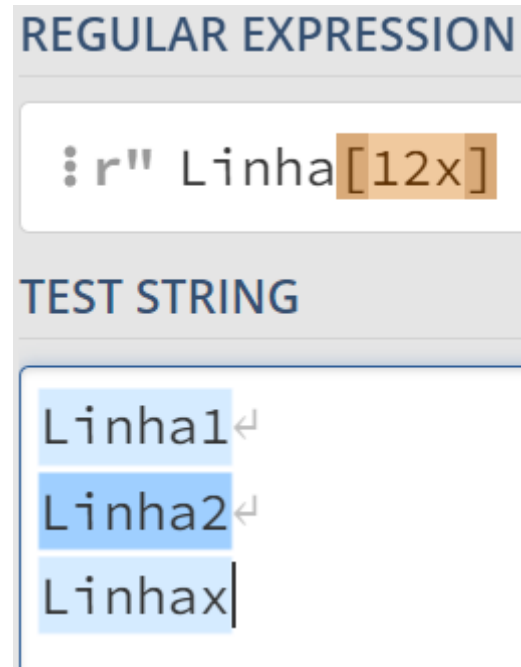
```
: r" [Ss][Ii][Mm]
```

TEST STRING

```
Sim•sim•sIm•sIM|
```

Classe de Caracteres

- Também podemos utilizar dígitos ou uma mistura de dígitos e outros caracteres para filtrar nossos dados utilizando classes:



regex – classes de caracteres

Usados para casar com um tipo de caracter específico.

Símbolo	Significado
.	Um caractere qualquer
[...]	Uma lista de caracteres
[^...]	Uma lista de caracteres proibidos
\s	Qualquer caractere branco/vazio
\S	Qualquer caractere não branco/vazio
\d	Qualquer dígito
\w	Qualquer caractere “palavra”

Exemplos:
mam[aã]o
\d\d
luc[^a]s

Negação de Classes

- Assim como podemos definir quais caracteres devem necessariamente aparecer, também podemos definir aqueles que **não devem** aparecer.
- Utilizamos “[^caracteres]” para isto.

```
REGULAR EXPRESSION
:r" [gr]ato

TEST STRING
gato, •pato, •rato
mato, •sapato, •trato
```

```
REGULAR EXPRESSION
:r" [^gr]ato

TEST STRING
gato, •pato, •rato
mato, •sapato, •trato
```

Exercício

REGULAR EXPRESSION

TEST STRING

bear•year•gear•fear•dear

REGULAR EXPRESSION

TEST STRING

bear•year•gear•fear•dear

Uso de Ranges

- E caso precisássemos considerar diversos caracteres, não apenas dois ou três?
- Precisamos definir todos de forma extensiva?
- Não!
- Podemos definir um range de caracteres utilizando o operador “-”.
- O padrão vai considerar como válido qualquer caractere entre o range, não apenas os caracteres presentes na ER.

Uso de Ranges

- Ranges numéricos:

```
REGULAR EXPRESSION 7 matches
:r" [1-7]teste
TEST STRING
1teste, 2teste, 3teste, 4teste
5teste, 6teste, 7teste, 8teste
```

- Ranges alfabéticos:

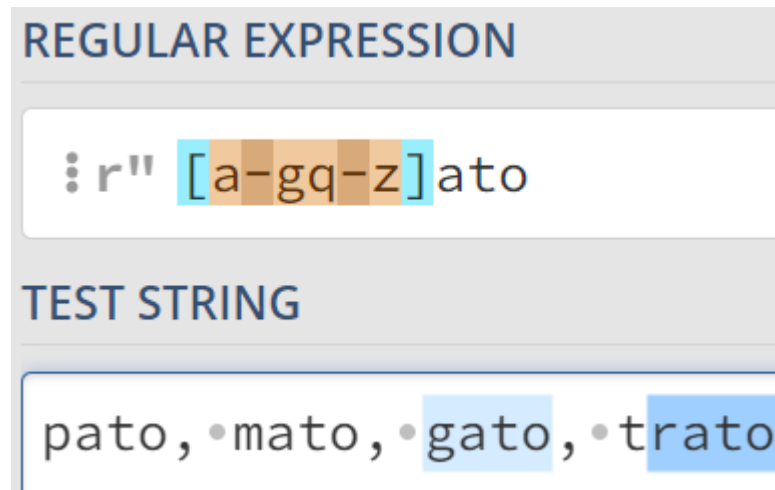
```
REGULAR EXPRESSION
:r" [a-z]ato|
TEST STRING
pato, mato, gato, trato
```

```
REGULAR EXPRESSION
:r" [a-h]ato
TEST STRING
pato, mato, gato, trato
```

```
REGULAR EXPRESSION
:r" [m-z]ato
TEST STRING
pato, mato, gato, trato
```

Uso de Ranges

- Também é possível utilizar múltiplos ranges não contínuos:



The image shows a web-based regular expression editor. It has two main sections: 'REGULAR EXPRESSION' and 'TEST STRING'. In the 'REGULAR EXPRESSION' section, the text ': r" [a-gq-z]ato' is entered. The characters 'a', 'g', 'q', and 'z' in the range '[a-gq-z]' are highlighted in orange, while the brackets and hyphen are in light blue. In the 'TEST STRING' section, the text 'pato, •mato, •gato, •trato' is entered. The words 'gato' and 'trato' are highlighted in light blue, while 'pato' and 'mato' are not. The bullet points '•' are used as separators between the words.

```
REGULAR EXPRESSION
:r" [a-gq-z]ato

TEST STRING
pato, •mato, •gato, •trato
```

- Para este caso o padrão apenas **aceita** letras de *a* até *g* (*a b c...g*) e de *q* até *z* (*q r s...z*) e **rejeita** letras após *g* e antes de *q* (*h i j..p*).

Exercício

REGULAR EXPRESSION

TEST STRING

bear•year•gear•fear•dear|

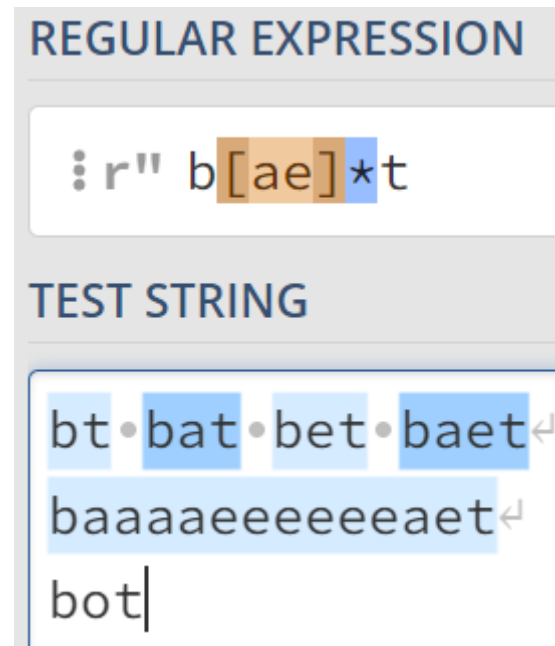
REGULAR EXPRESSION

TEST STRING

Lucas•Paulo•Maria•↵
teste•coisa•resto

Asterisco

- O * pode ser usado para definir que um caractere deve aparecer **zero ou mais** vezes em um texto.



Quantificadores

Usados para quantificar um padrão

Símbolo	Significado
?	zero ou um
*	zero um ou mais
+	um ou mais
{n,m}	De n até m ocorrências
*?	zero um ou mais (non-Greedy)
+	um ou mais (non-Greedy)

Exemplos:

0*

\d+

L{1,3}

Diferença entre quantificadores
gulosos e não gulosos (lazy):

Seja a string: "bcdabdc**ba**bcd"

$^{\wedge}(.*)ab$ bcdabdc**ba**bcd

$^{\wedge}(.*)?ab$ bcdabdc**ba**bcd

Exercício

REGULAR EXPRESSION

TEST STRING

Lucas•Paulo•Maria•
teste•coisa•resto

REGULAR EXPRESSION

TEST STRING

aaaabcc↵
aabbbbbc↵
aacc↵
a|

REGULAR EXPRESSION

TEST STRING

1•file•found↵
2•files•found↵
24•files•found↵
No•files•found↵

Outros Padrões Úteis

- Definir uma subexpressão, onde o conteúdo inteiro é verificado em conjunto:
 - (*<expressão>*)
 - Ex: *ab** aceitaria *a, ab, abb, abbb...*
*ab(ab)** aceitaria apenas *ab, abab, ababab...*
- Encontrar um número exato de vezes o padrão:
 - expressão{*m*}
 - Ex: *a{3}* aceita apenas *aaa* (ou textos maiores que contenham *aaa*)
- Encontrar um número de vezes entre *m* e *n* o padrão:
 - expressão{*m,n*}
 - Ex: *a{2,3}* aceita apenas *aa* ou *aaa* (ou textos maiores que contenham *aa* ou *aaa*)

regex – grupos

Usados para marcar grupos na regex (e também para facilitar o uso do ou |)

Símbolo	Significado
(...)	Grupo
	Ou

Exemplos:
(super|hiper)?mercado

regex – escape

Usado para transformar metacaracteres em literais

Símbolo	Significado
\?	Interrogação
\[Colchete
\.	ponto

Exemplos:

`www\[^\.]+\.`com

regex – olhar em volta (look around)

Usados para checar algum padrão antes ou depois do casamento.

Símbolo	Significado
(?=regex)	Executa a regex adiante para ver se casa com o texto a seguir.
(?!regex)	Executa a regex adiante para ver se não casa com o texto a seguir.
(?<=regex)	Executa a regex na string que veio antes para ver se casa com o texto anterior.
(?<!regex)	Executa a regex na string que veio antes para ver se não casa com o texto anterior.

Exemplos:

lucas (?=kupssinskü)

Exercício

REGULAR EXPRESSION

TEST STRING

```
<a href="www.google.com">nome•do•link</a>↵
```

```
<a>nome•do•link</a>
```

Exercício

REGULAR EXPRESSION

TEST STRING

```
<a href="www.google.com">nome•do•link</a>␣  
<strong>nome•do•link</strong>|
```

REGULAR EXPRESSION

12 matches (707 steps, 0.2)

"gi" |

TEST STRING

User-Agent:.*

Disallow: /jornalismo/g1/

Disallow: /_ssi/

Disallow: /teste-*.html\$

Disallow: /beta/

Disallow: /componentes/

Disallow: /busca/*

Disallow: /globo-news/jornal-globo-news/videos/v/globo-news-ao-vivo/61910/

Disallow: /globonews/playlist/globonews-ao-vivo.ghml

Disallow: *globo-cdn-src/*

Disallow: /zeta/

Disallow: /content-aggregator/

Disallow: /jogos-app/

Sitemap: https://g1.globo.com/sitemap/g1/sitemap.xml

Sitemap: https://g1.globo.com/sitemap/Apuração/g1/sitemap.xml

Guia Rápido de regex

<code>^</code>	Matches the beginning of a line
<code>\$</code>	Matches the end of the line
<code>.</code>	Matches any character
<code>\s</code>	Matches whitespace
<code>\S</code>	Matches any non-whitespace character
<code>*</code>	Repeats a character zero or more times
<code>*?</code>	Repeats a character zero or more times (non-greedy)
<code>+</code>	Repeats a character one or more times
<code>+?</code>	Repeats a character one or more times (non-greedy)
<code>[aeiou]</code>	Matches a single character in the listed set
<code>[^XYZ]</code>	Matches a single character not in the listed set
<code>[a-z0-9]</code>	The set of characters can include a range
<code>(</code>	Indicates where string extraction is to start
<code>)</code>	Indicates where string extraction is to end

Exercícios

- <https://regex101.com/>
- Construa expressões regulares que casem com:
 - CPF (###.###.###-##)
 - Data (dd/mm/yyyy) e (dd/mm/yy) e (dd-mm-aaaa) e (dd-mm-aa)
 - Conteúdo de tag loc de um sitemap.xml
 - Exemplo: “<loc>**http://192.168.1.105:8000/places/default/view/Aland-Islands-2**</loc>”

Leitura Indicada

- JARMUL, Katharine; LAWSON, Richard. **Python Web Scraping**. Packt Publishing Ltd, 2017.
 - Capítulo 1