Tutorial simples de Regex para iniciantes

O que é **Regex**? Regex, abreviação de "Expressão Regular", é uma sequência de caracteres que define um padrão de pesquisa. Pense nisso como um código especial que pode encontrar strings específicas dentro de um texto maior.

Por que usar Regex? Regex é usado na programação para:

- Procurando um padrão específico em um texto.
- Substituindo partes de um texto.
- Validar se uma string corresponde a um padrão (como verificar se uma entrada é um email válido).

Padrões Regex básicos:

```
 . : Corresponde a qualquer caractere, exceto uma nova linha.
\d : Corresponde a qualquer dígito (0-9).
\D: Corresponde a qualquer não dígito.
\w : Corresponde a qualquer caractere de palavra (a-z, A-Z, 0-9, _).
\W : Corresponde a qualquer caractere que não seja de palavra.
\s : Corresponde a qualquer espaço em branco (como espaço ou tabulação).
\S : Corresponde a qualquer espaço que não seja em branco.
 ^ : Corresponde ao início de uma string.
 $ : corresponde ao final de uma string.
```

Quantificadores:

```
- *: 0 ou mais
- +: 1 ou mais
- ? : 0 ou 1
- {n}: Exatamente n vezes
- {n,}: n ou mais vezes
- {n,m}: entre n e m vezes
```

Exemplo de Python:

Neste exemplo, estamos procurando uma ou mais ocorrências de caracteres de palavras. O padrão é: **"\w+"** (encontrar 1 ou mais caracteres). O método **findAll** do objeto **re** retorna uma lista de strings encontrados no texto:

```python
import re

# Encontre todas as palavras em um texto
text = "Hello, World!"
words = re.findall(r"\w+", text)
print(words)  # ['Hello', 'World']
```

Exemplo em Java:

```java
import java.util.regex.*;

public class RegexExample {
    public static void main(String[] args) {
        String text = "Hello, World!";

        // Encontre todas as palavras em um texto
        Pattern pattern = Pattern.compile("\\w+");
        Matcher matcher = pattern.matcher(text);
```

```
        while (matcher.find()) {
            System.out.println(matcher.group());
        }
        // Saída: Hello
        //         World
    }
}
```

Padrões com combinações

Vamos pensar em padrões mais complexos. Que tal validar um **email**? Como seria isso? Como podem ser os emails?

```
- fulano@teste.com
- fulano.detal@mail.teste.com.br
- c.wee.232@t.r.g.c
- fulano_de_tal@teste-mail.com
- +cleuton@teste.com
- cicrano@t.c
```

E o que seriam emails inválidos?

```
- beltrano
- 1.2.3.4
- beltrano@teste
- _g@_klj.com (tem underscore após o "@")
- *cleuton@teste.com
- cleuton@teste+1.com
- cicrano@t.
- cicrano@.t
- cicrano@.t.g
- cicrano@teste#r.com
- "  c@t.com" (tem espaço no início do string)
```

Então o email tem 3 partes com regras próprias:

1. Nome: Um ou mais caractere alfabético ou numérico incluindo: ".", "+", "-" e "_", mas não pode ter caracteres especiais ou whitespace (branco, CR, LF).
2. Arroba: Obrigatório.
3. Domínio: Um ou mais caracteres alfabéticos ou numéricos, uncluindo: "." e "-", precisa ter pelo menos um ponto precedido por um caractere e seguido por um caractere.

Como seria um **matcher** para isso?

Grupo nome: Começa no início do email, portanto, temos que usar o "^". Para este grupo, podemos usar o "\w", e se repete algumas vezes:

```
^\w+
```

Arroba:

```
@
```

Grupo domínio:

```
\w+
```

Então, será que esse padrão "^\w+@\w+" resolveria? Vejamos o teste com nossos exemplos de emails válidos:

```
% python email_validator.py
Enter an email address (or 'exit' to quit): fulano@teste.com
This is a valid email address.
Enter an email address (or 'exit' to quit): fulano.detal@mail.teste.com.br
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): fulano_de_tal@teste-mail.com
This is a valid email address.
Enter an email address (or 'exit' to quit): fulano_de_tal@teste-mail.com
This is a valid email address.
Enter an email address (or 'exit' to quit): +cleuton@teste.com
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): cicrano@t.c
This is a valid email address.
```

Temos alguns falsos negativos nesses testes. Agora, vejamos o teste com nossos exemplos de emails inválidos:

```
% python email_validator.py
Enter an email address (or 'exit' to quit): beltrano
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): 1.2.3.4
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): beltrano@teste
This is a valid email address.
Enter an email address (or 'exit' to quit): _g@_klj.com
This is a valid email address.
Enter an email address (or 'exit' to quit): *cleuton@teste.com
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): cleuton@teste+1.com
This is a valid email address.
Enter an email address (or 'exit' to quit): cicrano@t.
```

```
This is a valid email address.
Enter an email address (or 'exit' to quit): cicrano@.t
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): cicrano@.t.g
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): cicrano@teste#r.com
This is a valid email address.
Enter an email address (or 'exit' to quit):  c@t.com
This is NOT a valid email address.
```

Falhas no grupo **nome**:

1. Não aceita pontuação (fulano.detal@mail.teste.com.br).
2. Não aceita sinal "+" (+cleuton@teste.com).

Falhas no grupo **domínio**: 3. Aceitou domínio de uma só palavra (beltrano@teste). 4. Aceitou sinal de "+" no domínio (cleuton@teste+1.com). 5. Aceitou domínio terminando com "." (cicrano@t.). 6. Aceitou caracteres especiais no domínio (cicrano@teste#r.com). 7. Aceitou domínio terminando antes do final do string (beltrano@teste t).

E tem mais uma coisinha... Eu, como sou muito chato, testei isso: "beltrano@teste t", veja o resultado:

```
% python email_validator.py
Enter an email address (or 'exit' to quit): beltrano@teste t
This is a valid email address.
```

Repensando nossa **regex** para validar email, vimos que acertamos no início do string de email. Não pode haver espaço no início. Ok. Mas erramos no grupo **nome** pois estamos permitindo coisas demais e de menos. Podemos criar **classes de caracteres**, colocando os caracteres válidos entre colchetes. Por exemplo:

- [a-z]: Qualquer letra minúscula entre "a" e "z".
- [a-z0-9]: Qualquer letra minúscula entre "a" e "z" ou qualquer dígito entre 0 e 9.
- [a-z.+]: Qualquer letra minúscula entre "a" e "z", ponto ou "+".

Então podemos rever as regras e usar algo assim para o grupo **nome**, que deve ser sempre alinhado ao início do string:

```
^[a-zA-Z0-9_.+-]+
```

O sinal "+" no final indica uma ou mais ocorrências. E se pudesse ser zero ou mais? Como faríamos?

```
^[a-zA-Z0-9_.+-]*
```

E para zero ou um?

```
^[a-zA-Z0-9_.+-]?
```

Bom, a arroba está certa. Então vamos para o grupo **domínio**. A princípio, seria algo assim:

```
[a-zA-Z0-9-]+
```

Ok, mas faltou o ponto. Tem que ter pelo menos um ponto e uma continuação, então podemos fazer assim:

```
[a-zA-Z0-9-]+\.
```

Por que tivemos que usar a barra invertida? Caracteres especiais como ".", fora dos colchetes, exigem o uso de **escape character** para não serem confundidos com comandos **regex** (ponto significa "qualquer caractere", mas nós queremos apenas o ponto mesmo). Ok, mas e a continuação do domínio?

```
[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+
```

Temos um grupo de caracteres que se repete, um ponto e pelo menos outro grupo de caracteres que se repete, podendo ter ponto. Faltou só a informação que o domínio tem que ficar no final do string, não admitindo espaços. Então ficaria assim:

```
[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$
```

Nossa regex completa seria assim:

```
^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$
```

Ainda tem um furo nessa regex... Pode me dizer qual é? E se eu tentar:

```
Enter an email address (or 'exit' to quit): fulano.de.tal@teste.com.
This is a valid email address.
```

Não pode terminar em ponto... Como podemos resolver isso? Ah! Já sei! Será que tem um jeito de **negar** um caracter ao final?

Dentro de classes de caracteres, podemos começar com um "^" o que negaria a classe toda. Isso é problemático, pois fora da classe de caractere, o "^" serve para alinhar ao início do texto. Mas é assim

mesmo. Se você quiser forçar o caractere "^" dentro de colchetes, sem significar negação, então tem que precedê-lo com uma barra invertida. Vejamos:

- a : O caractere "a"
- [^a] : Qualquer caractere sem ser o "a"

Vamos tentar fazer algo assim:

```
^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+[^.]$
```

Agora, nosso email não pode terminar com ponto! Vejamos alguns test cases:

```
 % python email_validator.py
Enter an email address (or 'exit' to quit): fulano@teste.com
This is a valid email address.
Enter an email address (or 'exit' to quit): fulano.detal@mail.teste.com.br
This is NOT a valid email address.
```

Falhou logo no segundo... Por que? Use a imaginação! Vamos lá:

```
fulano.detal@mail.teste.com.br
            ^   ^^    ^
            |   ||    |
            1   23    4
```

Na posição 1 temos um grupo de caracteres começando. Na posição 2, temos um ponto. Na posição 3 temos outro grupo de caracteres começando. E na posição 4 temos um ponto, o que ele não vai permitir.

Grupos

Em regex temos o conceito de grupos de caracteres. Um grupo fica entre parêntesis. Podemos aplicar repetição ao grupo inteiro. Se pensarmos no segundo componente do **domínio** como um grupo que se repete?

- nome: "fulano.detal"
- arroba: "@"
- domínio: "mail"
- outro: ".teste"
- mais outro: ".com"
- e finalmente: ".br"

Podemos dividir a parte do **domínio** acrescentando um grupo que deve terminar o email:

```
^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)+$
```

Calma que eu vou explicar... Depois da arroba, temos uma classe de caracteres que se repete uma ou mais vezes. Depois, temos um ponto e depois, temos um grupo de caracteres que se repete uma ou mais vezes: "(.[a-zA-Z0-9-]+)+$". Este grupo começa com um ponto, pode se repetir, e deve encerrar o string.

Agora, todos os nossos **test cases** passaram:

```
 % python email_validator.py
Enter an email address (or 'exit' to quit): fulano@teste.com
This is a valid email address.
Enter an email address (or 'exit' to quit): fulano.detal@mail.teste.com.br
This is a valid email address.
Enter an email address (or 'exit' to quit): c.wee.232@t.r.g.c
This is a valid email address.
Enter an email address (or 'exit' to quit): fulano_de_tal@teste-mail.com
This is a valid email address.
Enter an email address (or 'exit' to quit): +cleuton@teste.com
This is a valid email address.
Enter an email address (or 'exit' to quit): cicrano@t.c
This is a valid email address.
Enter an email address (or 'exit' to quit): beltrano
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): 1.2.3.4
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): beltrano@teste
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): _g@_klj.com
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): *cleuton@teste.com
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): cleuton@teste+1.com
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): cicrano@t.
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): cicrano@.t
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): cicrano@.t.g
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): cicrano@teste#r.com
This is NOT a valid email address.
Enter an email address (or 'exit' to quit):   c@t.com
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): beltrano@teste t
This is NOT a valid email address.
Enter an email address (or 'exit' to quit): fulano@.com
This is NOT a valid email address.
Enter an email address (or 'exit' to quit):
fulano.detal@mail.teste.com.br.
This is NOT a valid email address.
```

No arquivo **email_validator.py** você tem a implementação **Python** desse validador de email, e no arquivo **EmailValidator.java** tem a versão **Java**.

Dicas para iniciantes:

- Comece com padrões simples e depois combine-os.
- Sempre teste seus padrões de regex usando pequenos exemplos.
- Existem muitas ferramentas online, como regex101.com, que podem ajudá-lo a testar e compreender seus padrões.
- Lembre-se de que regex pode ser complexo, mas com prática você pegará o jeito!
- Conclusão: Regex é uma ferramenta poderosa para manipulação de strings em programação. Com um pouco de prática, você pode usá-lo para pesquisar, substituir e validar strings facilmente em Python e Java.