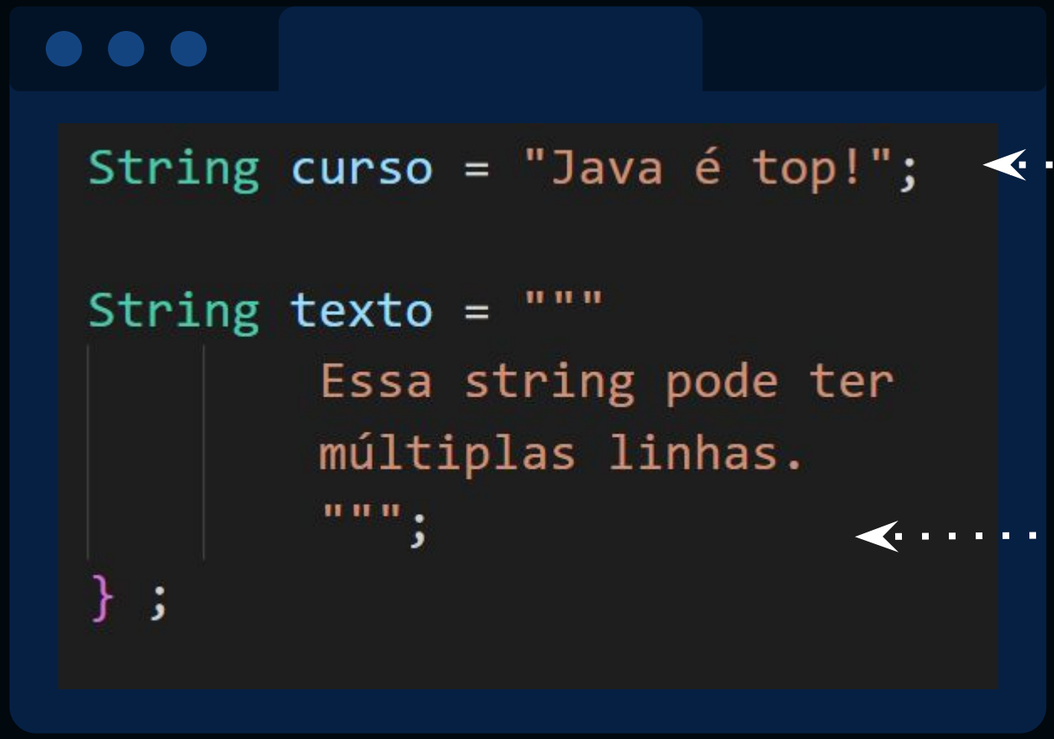
# Strings e Regex

# Strings

Strings são **sequências de caracteres** usadas para **representar texto**, delimitadas por aspas duplas ou triplas.




Aspas duplas são usadas para uma atribuição comum.

Aspas triplas permitem criar strings de múltiplas linhas com quebras de linha.
Esse recurso é chamado de **Text Block**

# Métodos de manipulação

Os métodos de manipulação de strings são usados para transformar, modificar ou analisar textos.

| Método | Conceito | Exemplo | Saída |
|---|---|---|---|
| trim() | Remove espaços em branco (ou caracteres especificados) do início e fim da string. | "  exemplo  ".trim() | "exemplo" |
| toLowerCase() | Converte todos os caracteres da string para letras minúsculas. | "EXEMPLO".toLowerCase() | "exemplo" |
| toUpperCase() | Converte todos os caracteres da string para letras maiúsculas. | "exemplo".toUpperCase() | "EXEMPLO" |
| replace() | Substitui uma substring específica por outra na string. | "Olá Mundo".replace("Mundo", "Java!") | "Olá Java!" |

# String.format()

O String.format() é um método da classe String que  permite a formatação de strings de forma simples e legível, incorporando expressões e variáveis diretamente no texto.

```java
String estudante = "Pedro";
int nota = 10;
String mensagem = String.format("O estudante %s obteve a nota %d", estudante, nota);

System.out.println(mensagem);
```

A mensagem impressa será a string “O estudante Pedro obteve nota 10”.

# printf

O System.out.printf() também pode ser usado para formatar uma saída de texto.

```
String estudante = "Pedro";
int nota = 10;

System.out.printf("O estudante %s obteve a nota %d", estudante, nota);
```

A mensagem impressa será a string “O estudante Pedro obteve nota 10”.

# Caracteres de formatação

Caracteres que usamos prefixados com o símbolo de percentual % para ser substituído no texto por um valor.

| Símbolo | Descrição | Exemplo | Saída |
|---|---|---|---|
| %s | Substitui uma string | String.format(“Olá %s”, variável) | “Olá João” |
| %d | Substitui um inteiro | String.format(“%d anos”, variável) | “20 anos” |
| %f | Substitui um float/double | String.format(“Nota %f”, variável) | “Nota 9,500000” |
| %n | Insere uma nova linha | String.format(“Oi! %n turma”) | “Oi<br>turma” |
| %.2f | Formata o valor com duas casas decimais | String.format(“Nota %f”, variável) | “Nota 9,50” |

# Indexação de strings com charAt

A indexação permite acessar caracteres individuais de uma string através de seu índice, começando de 0 para o primeiro caractere.

```
String texto = "Alura";

System.out.println(texto.charAt(3));
```

Imprime o último 4º caractere da string que é “r”.

# Método substring

O substring permite extrair uma parte da string.

A sintaxe pode ser de duas formas. A primeira é `string(início, fim)`, onde:

- ✔ **início:** índice inicial. O caractere dessa posição é incluído na string final.
- ✔ **fim:** índice final. O caractere dessa posição é excluído na string final.

A segunda forma é `string(início)`, onde:

- ✔ **início:** índice a partir de qual o texto será extraído. O caractere dessa posição é incluído na string final e todos os outros subsequentes também.

# Exemplo

```
String texto = "Eu gosto de aprender coisas novas";

System.out.println(texto.substring(3, 20));

System.out.println(texto.substring(21));
```

Extrai os caracteres da posição 3 até a 20, resultando em 'gosto de aprender'.

Extrai os caracteres a partir da posição 21, resultando em “coisas novas”.

# Método contains()

O método contains() verifica se uma substring está presente em uma string.
Ele retorna **True** se a substring estiver presente na string e **False** caso contrário.
A verificação é *case sensitive*, ou seja, faz diferença entre maiúsculas e minúsculas.

```
String texto = "Eu gosto de aprender coisas novas";

System.out.println(texto.contains("gosto") );

System.out.println(texto.contains("Java"));
```

Verifica se a substring "gosto" está presente no conteúdo da variável texto, retornando **True** porque ela é encontrada na string.

Verifica se a substring "Java" está presente no conteúdo da variável texto, retornando False porque não está presente.

# Método startsWith()

O método startsWith() verifica se a string começa com uma substring específica.

Ele retorna **True** se a string iniciar com a substring especificada e **False** caso contrário.

```java
String texto = "Eu gosto de Java";

System.out.println(texto.startsWith("Eu") );

System.out.println(texto.startsWith("eu"));
```

Verifica se a string "Eu gosto de Java" começa com a substring "Eu", retornando **True** no primeiro println. No segundo println verifica se começa com "eu", retornando **False** devido à diferença de maiúsculas e minúsculas.

# Método endsWith()

O método endsWith() verifica se a string termina com uma substring específica.

Ele retorna **True** se a string terminar com a substring especificada e **False** caso contrário.

```
String texto = "Eu gosto de Java";

System.out.println(texto.endsWith("Java") );

System.out.println(texto.endsWith("java"));
```

Verifica se a string "Eu gosto de Java" termina com a substring "Java", retornando **True** no primeiro println. No segundo println verifica se termina com "java", retornando **False** devido à diferença de maiúsculas e minúsculas.

# Regex

Regex (expressões regulares) é uma ferramenta poderosa para buscar, manipular e validar padrões em strings usando uma linguagem de padrões.

É essencial para lidar com padrões de texto mais complexos, como e-mails, números de telefone, CPF, validação de entradas, entre outros.

Com Regex é possível criar padrões dinâmicos para validar, buscar ou substituir dados em textos, de forma compacta e poderosa para lidar com manipulação de strings.

Ele é baseada na API **java.util.regex**, que contém as classes **Pattern** e **Matcher**

# Caracteres literais

Caracteres literais em regex correspondem exatamente aos caracteres na string de entrada, como letras, números e símbolos, sem nenhum comportamento especial, a menos que sejam caracteres especiais.

Por exemplo, se você usar o caractere a em uma regex, ele corresponderá ao caractere "a" em uma string, sem nenhuma modificação ou interpretação especial.

Exemplo: Se você quiser encontrar a palavra "Java" em um texto, pode usar a regex Java.

Caracteres especiais têm um significado específico dentro das expressões regulares. Para serem usados como caracteres literais devem ser "escapados" com uma barra invertida (\).

# Caracteres especiais

Caracteres especiais em expressões regulares são símbolos com significados específicos que permitem definir padrões complexos.

| Símbolo | Descrição |
|---|---|
| . | Corresponde a qualquer caractere, exceto quebra de linha |
| \d | Corresponde a qualquer dígito (0-9) |
| \D | Corresponde a qualquer caractere que não seja um dígito |
| \w | Corresponde a qualquer caractere alfanumérico (letras, números e underline) |
| \W | Corresponde a qualquer caractere que não seja alfanumérico |
| \s | Corresponde a qualquer espaço em branco (espaço, tabulação, etc.) |
| \S | Corresponde a qualquer caractere que não seja espaço em branco |
| ^ | Início da string |
| $ | Fim da string |

# Classe de caracteres

Classes de caracteres em expressões regulares são grupos de caracteres definidos entre colchetes, que pode corresponder a diferentes padrões.

| Símbolo | Descrição |
|---|---|
| [abc] | Corresponde a qualquer caractere dentro dos colchetes, exemplo 'a', 'b' ou 'c' |
| [^abc] | Corresponde a qualquer caractere exceto os que não estejam dentro dos colchetes |
| [a-z] | Corresponde a qualquer caractere de 'a' a 'z' (minúsculas) |
| [A-Z] | Corresponde a qualquer caractere de 'A' a 'Z' (maiúsculas) |
| [0-9] | Corresponde a qualquer dígito (0-9) |
| [a-zA-Z] | Corresponde a qualquer letra, maiúscula ou minúscula |

# Quantificadores

Quantificadores em expressões regulares são usados para especificar o número de ocorrências de um padrão.

| Símbolo | Descrição |
|---|---|
| * | Corresponde a 0 ou mais ocorrências do padrão anterior |
| + | Corresponde a 1 ou mais ocorrências do padrão anterior |
| ? | Corresponde a 0 ou 1 ocorrência do padrão anterior |
| {n} | Corresponde exatamente a n ocorrências do padrão anterior |
| {n,} | Corresponde a n ou mais ocorrências do padrão anterior |
| {n,m} | Corresponde entre n e m ocorrências do padrão anterior |

# Exemplo

Regex que corresponde aos números de telefone no formato (XX) XXXX-XXXX ou (XX) XXXXX-XXXX, como os telefones "(12) 3456-7890" e "(21) 98765-4321".

Regex: `\\(\\d{2}\\) \\d{4}-\\d{4}`

Explicação:

- ✔ **\\(\\d{2}\\):** Dois dígitos dentro de parênteses, como o código de área.
- ✔ **\\d{4}:** Quatro dígitos para o início do número do telefone.
- ✔ **-:** O hífen literal.
- ✔ **\\d{4}:** Quatro dígitos no final.

# Exemplo

Regex que corresponde  a datas no formato "DD/MM/AAAA", como as datas "12/05/2023" e "01/01/2021"

Regex: `\\b\\d{2}/\\d{2}/\\d{4}\\b`

Explicação:

- ✔ **\\b:** Limite de palavra para garantir que a correspondência seja uma data completa.
- ✔ **\\d{2}:** Dois dígitos para o dia e o mês.
- ✔ **/:** O caractere de barra literal.
- ✔ **\\d{4}:** Quatro dígitos para o ano.
- ✔ **\b:** Limite de palavra no final para garantir que não haja caracteres extras após a data.

# Métodos das classes Matcher

A API oferece métodos para facilitar a interação com textos de maneira flexível e eficiente.

| Método | Descrição | Exemplo de uso | Saída |
|---|---|---|---|
| find | Procura por um padrão em qualquer parte da string. Retorna o primeiro resultado encontrado. | matcher.find("\\d+", "Há 1234 alunos") | Retorna '1234' |
| matches | Verifica se a string inteira corresponde ao padrão | String texto = “estudando java” matcher.matches( "java") | Retorna booleano se houve um match |
| lookingAt | Verifica se a string parcialmente corresponde ao padrão | String texto = “estudando java” matcher.lookingAt() | Retorna booleano se houve um match |
| replaceAll | Substitui ocorrências do padrão por uma string. | Pattern pattern = Pattern.compile("\\d"); matcher.replaceAll(“#”) | Troca todos os números do texto por # |

# Método group

O método group() é utilizado para acessar a correspondência encontrada por uma expressão regular, retornando o texto correspondente ao padrão.

Ele é comum quando se usa métodos como matcher.find() que retornam um objeto de correspondência.

Se uma correspondência for encontrada, group() retorna o valor da string que corresponde ao padrão.

# Exemplo

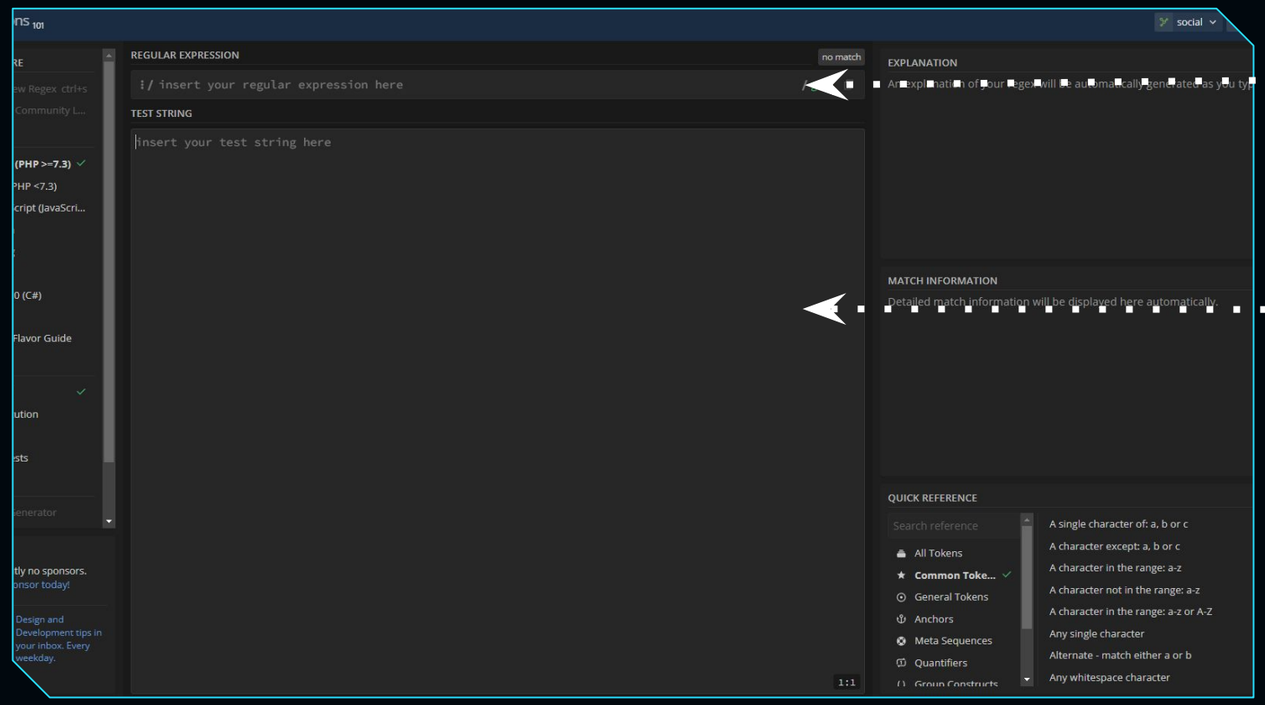
O código a seguir utiliza os métodos find() e group() para extrair um endereço de email de uma string:

```
String texto = "Meu email é exemplo@gmail.com";
Pattern pattern = Pattern.compile("(\\w+)@(\\w+\\.\\w+)"); // Captura email
Matcher matcher = pattern.matcher(texto);

if (matcher.find()) {
    System.out.println("Email completo: " + matcher.group());   // grupo 0 (tudo)
    System.out.println("Usuário: " + matcher.group(1));         // grupo 1 (antes do @)
    System.out.println("Domínio: " + matcher.group(2));         // grupo 2 (depois do @)
    }
}
```

# Ferramentas

Testar expressões regulares em ferramentas como o regex101 é importante para validar e depurar padrões antes de usá-los no código.



# Para se aprofundar em regex

Programação > Cursos de Computação

Curso de

**Expressões Regulares: faça buscas, validações e substituições de textos**

Acessar!

alura

Compartilhe um resumo de seus novos conhecimentos em suas redes sociais.

**#aprendizadoalura**