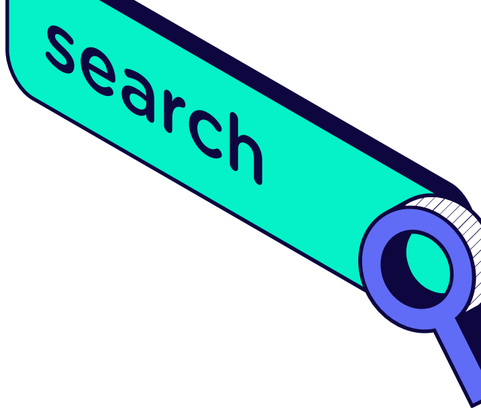
Power Up

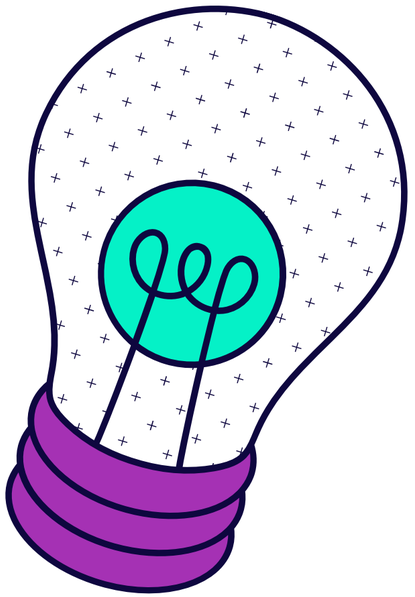
Introdução ao JavaScript

BYTES4FUTURE

FUNDAÇÃO AGA KHAN
comunidades em ação
LISBOA
área metropolitana de lisboa
PRR
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU

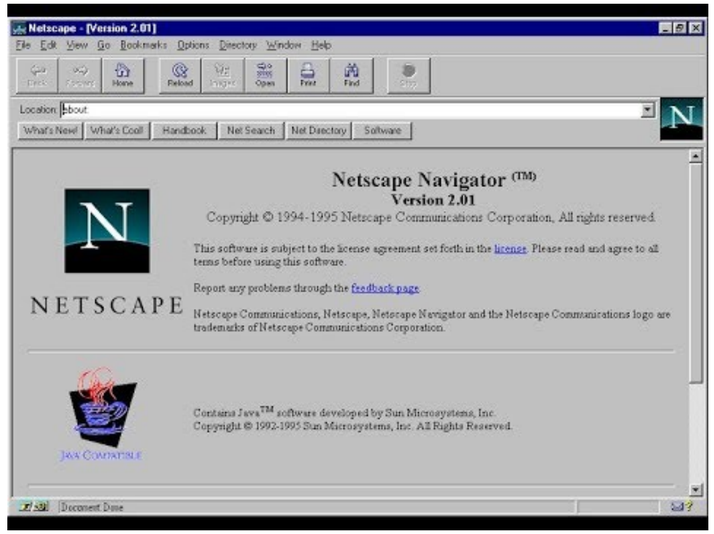
CONTEXTUALIZAÇÃO





História do Javascript

- Foi criado em 1995 pela Netscape Communications Corporation.
- Inicialmente, era conhecido como "LiveScript" e posteriormente foi renomeado para "**JavaScript**".
- Em 1997, o JavaScript foi padronizado pela ECMA International, resultando no ECMAScript, uma especificação que serve como base para o JavaScript moderno.
- Essa padronização permitiu que o JavaScript se tornasse uma linguagem mais amplamente aceita e utilizada em diferentes navegadores.



JavaScript

- JavaScript é uma linguagem de programação de alto nível amplamente usada para desenvolvimento web para adicionar **interatividade** e **dinamismo** a sites e aplicativos da web.

Hello World em JavaScript

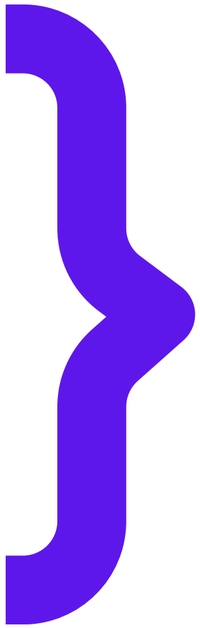
```
console.log("Olá Bytes4Future!");
```

Abrir um console no navegador

- Abrir Google Chrome
- "Pressione F12 no teclado."

  "Ou clique com o botão direito em qualquer lugar da página e selecione 'Inspecionar'."
- "Vá para a guia 'Console'."
- Execute:

```
console.log("Hello Word!");
```



VSCODE

Como criar um script

Há duas formas de adicionar Javascript ao ficheiro:

É sempre colocado no final do body!

Inline

```html
<html>
<head>
    <title>Exemplo de Script Inline</title>
</head>
<body>
    <h1>Exemplo de Script Inline</h1>

    <script>
        // Código JavaScript inline
        let mensagem = "Olá, Mundo!";
        console.log(mensagem);
    </script>
</body>
</html>
```

Externo

```html
<html>
<head>
    <title>Minha Página</title>
</head>
<body>
    <h1>Minha Página</h1>
    <script src="script.js"></script>
</body>
</html>
```

Como criar um ficheiro Javascript

1. Abre o VsCode
2. Cria um Novo Arquivo e guarda o ficheiro com extensão **.js**
3. Começa a escrever o código

Por exemplo:

```
console.log("Hello Word!");
```

- Para executar o código na terminal, abre o terminal no VSCode. (**ctrl + ç**)
- Executa o código colocando: **node + nome do arquivo**

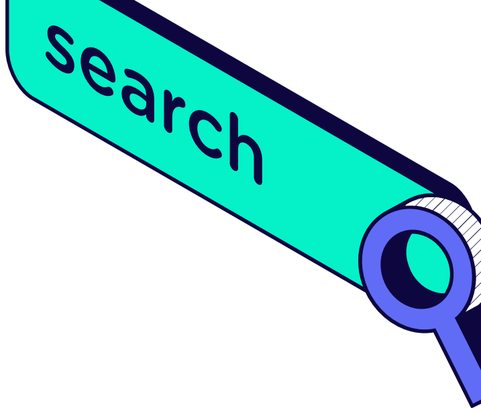
Como comentar em JS

Comentários de uma única linha

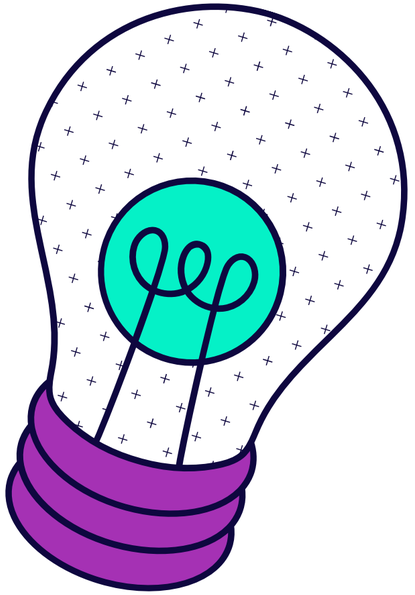
```
// Escreve o teu comentário aqui
```

Comentário de multiplas linhas:

```
/* Escreve o teu
   comentário
     aqui */
```



VÁRIAVEIS



Variáveis

São usadas para armazenar dados e valores.
Elas servem como caixas, que podem conter diferentes tipos de informações.

Para criar uma variável em JavaScript, pode usar as palavras-chave var, let ou const, seguidas pelo nome da variável.

Variáveis

- Variáveis são nomes que guardam um valor que pode ser alterado.
- São declaradas com a palavra reservada var ou let. **Dá preferência ao let.**

```
let aMinhaVariavel = 1;
let outraVariavel = "Teste";
```

Como Dar Nomes a Variáveis

- As variáveis costumam seguir a regra **camelCase**, ou seja, começam com minúsculas e a primeira letra de cada nova palavra é maiúscula.

```
let gradeAverage = 0
let totalScore = 9000
```

Variáveis

As variáveis em JavaScript não estão diretamente associadas a nenhum tipo de valor específico, e qualquer variável pode receber (e reatribuir) valores de todos os tipos:

let exemplo = 42; *// a variável agora é de tipo number*
exemplo = "bar"; *// exemplo agora é de tipo string*
exemplo = true; *// agora é um boolean*

Constantes

- Constantes são variáveis que guardam um valor que não pode ser alterado. São declaradas com a palavra reservada **const**.

```
const A_MINHA_CONSTANTE = 1;
const OUTRA_CONSTANTE = "Teste";
```

As constantes servem para guardar um valor que sabemos antes da execução do código e que não vai mudar.

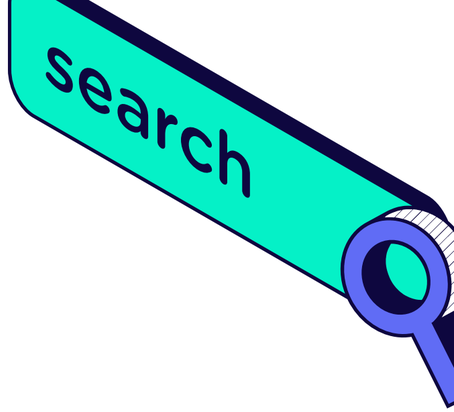
Principalmente quando este valor é utilizado em muitos pontos do código.

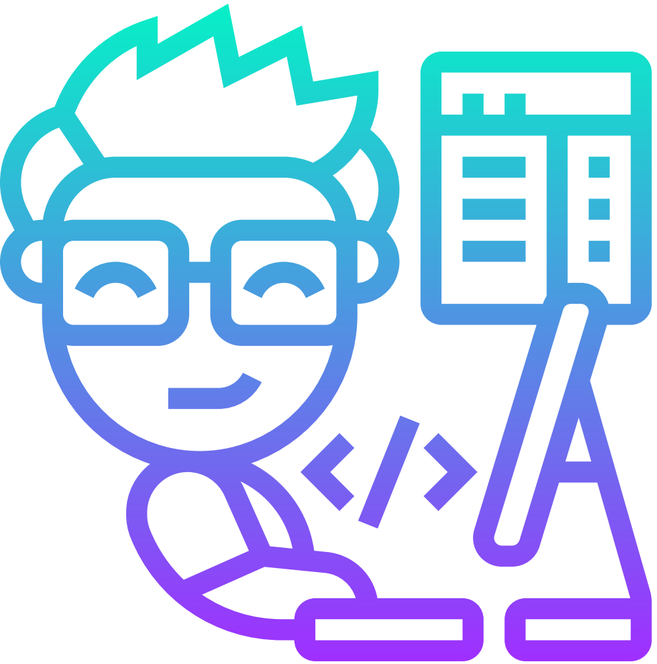
Como Dar Nomes a Constantes

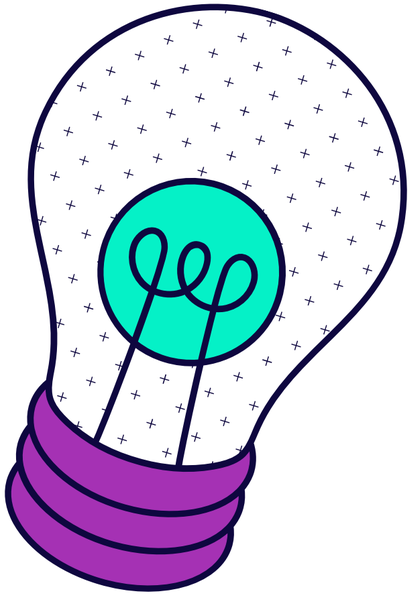
Já as constantes devem ter nomes em MAIÚSCULAS, separados por _.

```
const MAX_NUMBER = 100
const MIN_NUMBER = 0
```

QUIZ !!





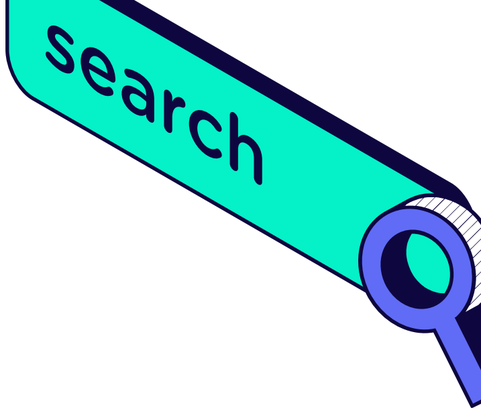


Vamos praticar

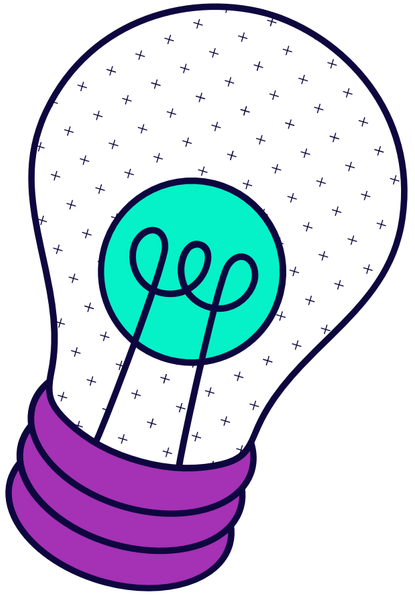
Exercício:

Escreve o código para declarar uma variável e uma constante, seguindo as boas praticas de **“Como dar nomes a Variaveis”.**

Imprime as variáveis na terminal.



ESTRUTURA DE DADOS



Tipos de Dados

- Apesar do JavaScript ser dinamicamente tipificado, é importante sabermos que existem tipos de dados, mas que não são expressamente escritos em código, como acontece em outras linguagens.

Tipos de Dados Primitivos

Existem em JavaScript 6 tipos de dados primitivos:

- **Number**
- **String**
- **Boolean**
- **Undefined:**
- **Symbol**
- **BigInt**

Number

Number

Representa números inteiros(**Integer**) ou de ponto flutuante(**Float**), como **42** ou **3.14**.

```
let n = 42
```

String

String

Tipo utilizado para representar dados **textuais**.
A string pode ser rodeada por ' ou ".

Uma característica das strings em JavaScript é que uma vez criadas estas não podem ser alteradas, apenas substituídas.

```
"Olá Mundo"
'Olá Mundo'
```

String

Existem algumas sequências de escape que deves conhecer:

- \n: Quebra de linha
- \t: Tabulação Horizontal
- \0: Caracter Nulo
- \': Apóstrofo
- \": Aspas
- \\: Barra inclinada para a esquerda

String

Exemplo:

```
let quebraLinha = "Primeira linha\nSegunda linha";
console.log(quebraLinha);
// Resultado:
// Primeira linha
// Segunda linha

let tabulacao = "Texto\tcom\ttabulação";
console.log(tabulacao);
// Resultado: "Texto    com    tabulação"

let aspasSimples = 'Isto é um apóstrofo: \'';
console.log(aspasSimples);
// Resultado: "Isto é um apóstrofo: '"

let aspasDuplas = "Isso são aspas duplas: \"";
console.log(aspasDuplas);
// Resultado: "Isso são aspas duplas: \""

let barraInclinada = "Uma barra inclinada para a esquerda: \\";
console.log(barraInclinada);
// Resultado: "Uma barra inclinada para a esquerda: \"
```

String Templates

São uma forma de criar strings que permitem incorporar variáveis e expressões diretamente em seu conteúdo.

Se quiseres definir uma string literal que tenha várias linhas, podes utilizar ` em vez de ".

```
const name = "Alice";
const age = 30;
const message = `Hello, my name is ${name} and I am ${age} years old.`;
```

Boolean

Boolean

- Tem apenas dois valores possíveis que significa que representam a verdade e a mentira, respectivamente.

```
true  // Verdadeiro
false // Falso
```

Undefined

Undefined

- Tem apenas um valor possível que significa que o nome não tem nenhum valor atribuído, ou seja, está indefinido

Uma variável à qual ainda não foi atribuído um valor tem o valor undefined.

Quando convertido para Boolean, é equivalente a false.

```
let myVar;
console.log(myVar); // Output: undefined
```

BigInt

BigInt

Tipo utilizado para representar números inteiros com qualquer dimensão.
Para definir que um valor deve ser tratado como BigInt em vez de Number, basta colocar um n depois do valor.

```
1n
9007199254740991n
BigInt(1)
```

Vamos praticar

Exercício:

Escreve o código para declarar três variáveis:

1. uma para armazenar um número inteiro,
2. outra para armazenar um número decimal e
3. uma terceira para armazenar uma string.

Imprime na terminal.
Utilize o **typeof** para verificar o tipo de variável que foi declarada.

OPERADORES

Operadores Aritméticos

Usados para realizar operações matemáticas.

- **+** (adição)
- **-** (subtração)
- * (multiplicação)
- **/** (divisão)
- **%** (módulo - resto da divisão)
- ** (exponenciação)

```
let a = 5;
let b = 3;
let soma = a + b; // soma é 8
let produto = a * b; // produto é 15
```

Operadores de Atribuição

Usados para atribuir valores a variáveis.

- **=** (atribuição)
- **+=** (atribuição de adição)
- **-=** (atribuição de subtração)
- ***=** (atribuição de multiplicação)
- **/=** (atribuição de divisão)
- **%=** (atribuição de módulo)
- ****=** (atribuição de exponenciação)

```
let x = 10;
x += 5; // x agora é 15
```

+

O operador **+** funciona como a soma.

O operador **+=** funciona da mesma forma, mas atribui ao lado esquerdo o resultado.

```
let a = 1;
let b = 2;
a + b // 3
a += b // 3. a = 3
a = a + b // 3. a = 3
```

-

O operador **-** funciona como a subtração.

O operador **-=** funciona da mesma forma, mas atribui ao lado esquerdo o resultado.

```
let a = 1;
let b = 2;
a - b // -1
a -= b // -1. a = -1
a = a - b // -1. a = -1
```

*

O operador * funciona como a multiplicação.

O operador ***=** funciona da mesma forma, mas atribui ao lado esquerdo o resultado.

```
let a = 5;
let b = 2;
a * b // 10
a *= b // 10. a = 10
a = a * b // 10. a = 10
```

/

O operador **/** funciona como a divisão.

O operador **/=** funciona da mesma forma, mas atribui ao lado esquerdo o resultado.

```
let a = 5;
let b = 2;
a / b // 2.5
a /= b // 2.5. a = 2.5
a = a / b // 2.5. a = 2.5
```

%

O operador **%** funciona como a resto da divisão inteira.

O operador **%=** funciona da mesma forma, mas atribui ao lado esquerdo o resultado.

```
let a = 5;
let b = 2;
a % b // 1
a %= b // 1. a = 1
a = a % b // 1. a = 1
```

**

O operador ** funciona como a potência.

O operador **= funciona da mesma forma, mas atribui ao lado esquerdo o resultado.

```
let a = 5;
let b = 2;
a ** b // 25
a **= b // 25. a = 25
a = a ** b // 25. a = 25
```

Operadores do tipo BigInt

Os operadores do tipo BigInt são iguais aos do tipo Number com a exceção da divisão /.

Isto porque o resultado passa a ser obrigatoriamente um **inteiro**, sendo as casas decimais ignoradas.

```
let a = 5n;
let b = 2n;
a / b // 2n
```

Operadores de Incremento e Decremento

Usados para aumentar ou diminuir o valor de uma variável em 1.

- **++** (incremento)
- **--** (decremento)

```
let contador = 5;
contador++; // contador é 6
```

Operadores de Comparação

Usados para comparar valores e produzir valores booleanos (true ou false).

- == (igual a)
- === (igual a e do mesmo tipo)
- != (diferente de)
- !== (diferente de ou tipo diferente)
- < (menor que)
- > (maior que)
- <= (menor ou igual a)
- >= (maior ou igual a)

```
let idade = 25;
let maiorDeIdade = idade >= 18; // true
```

== e ===

Os operadores de igualdade servem para comparar dois valores, retornando true se forem considerados iguais e false caso contrário.

A diferença entre o == e o === é o nível de igualdade necessário. O operador === só retorna true se tanto o valor como o tipo forem iguais.

```
"1" == 1 // true
1 == 1 // true
"1" === 1 // false
1 === 1 // true
```

!= e !==

Os operadores de diferença servem para comparar dois valores, retornando true se forem considerados diferentes e false caso contrário.

A diferença entre o != e o !== é o nível de diferença necessário. O operador != retorna true se tanto o valor como o tipo forem diferentes.

```
"1" != 1 // false
1 != 1 // false
2 != 1 // true
"1" !== 1 // true
1 !== 1 // false
```

> e >=

Os operadores > e >= têm o resultado true se o valor do lado esquerdo do operador for maior ou maior ou igual ao valor do lado direito e false caso contrário.

```
"a" > "A" // true
1 > 1 // false
1 > "1" // false

"a" >= "A" // true
1 >= 1 // true
1 >= "1" // true
```

< e <=

Os operadores < e <= têm o resultado true se o valor do lado esquerdo do operador for menor ou menor ou igual ao valor do lado direito e false caso contrário.

```
"a" < "A" // false
1 < 1 // false
1 < "1" // false

"a" <= "A" // false
1 <= 1 // true
1 <= "1" // true
```

Operadores Lógicos

Usados para realizar operações lógicas em valores booleanos.

- && (E lógico)
- || (OU lógico)
- ! (NÃO lógico)

```
let diaEnsolarado = true;
let vaiPassear = true;

let resultado = diaEnsolarado && vaiPassear;

console.log(resultado); // O resultado será true
```

```
let diaEnsolarado = false;
let vaiPassear = true;

let resultado = diaEnsolarado && vaiPassear;

console.log(resultado); // O resultado será false
```

&& (E)

- O operador **&&** serve para verificar se dois (ou mais) valores são simultaneamente **true**.

```
false && false // false
true && false // false
true && true // true
```

|| (OU)

- O operador || serve para verificar se pelo menos **um dos valores é true**.

- Se a expressão que está do lado esquerdo for true, a do lado direito pode ser ignorada.

```
false || false // false
true || false // true
true || true // true
```

! (INVERTE)

- O operador **!** (NÃO lógico) serve para negar um valor booleano. Ele **inverte** true para false e false para true..

```
!false // true
!true // false
```

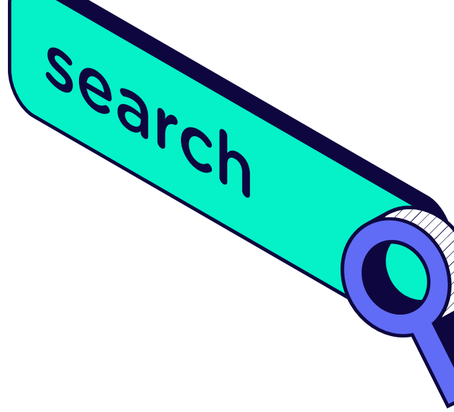
Operadores de Concatenação

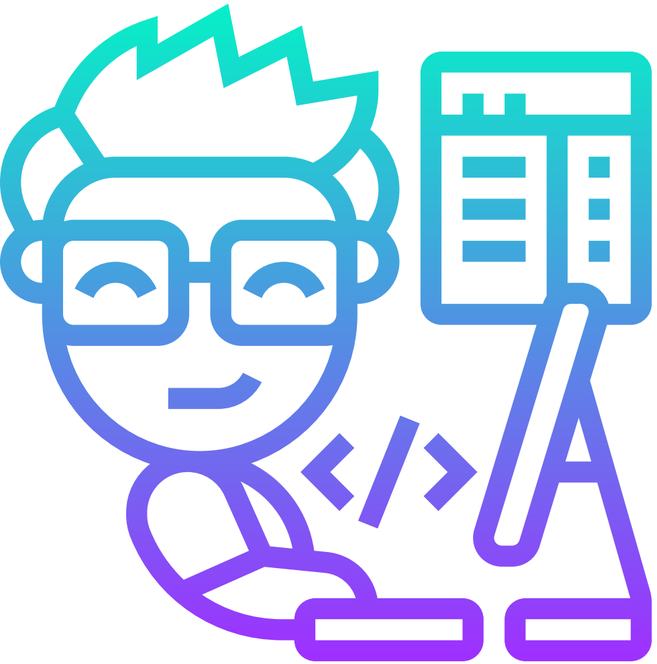
Usados para combinar strings.

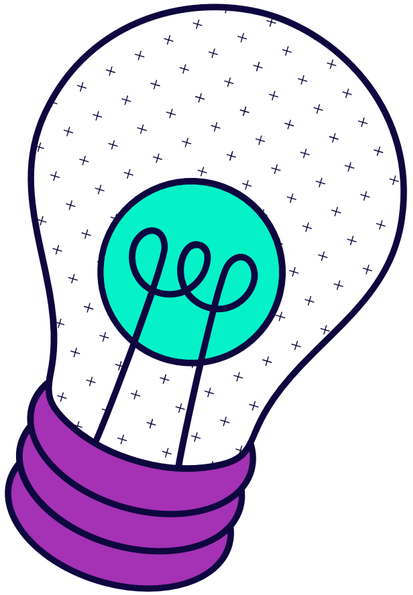
- + (concatenação de strings)

```
let nome = "Alice";
let sobrenome = "Smith";
let nomeCompleto = nome + " " + sobrenome; // "Alice Smith"
```

QUIZ !!







Vamos praticar

Exercício:

Faz a concatenação de duas strings que se encontram nas variáveis a e b, deixando um espaço entre elas e atribui o resultado à variável c.

let a = "Olá"
let b = "Mundo"

Exercício:

Cria duas variáveis, uma com um nome e uma idade e retorne uma string no formato abaixo:

"O meu nome é " " e tenho " ""

Exercício:

Soma o valor numérico das duas variáveis.
Guarda o resultado numa variável chamada soma.

```
// Aqui é preciso somar dois números
// Mas em que um deles é uma string
// Guarda o resultado da soma na variável soma

let num = 100
let str = “100"
```

Exercício:

Cálculo da Média de Duas Notas:

1. Declara duas variáveis para armazenar as notas, atribuindo valores predefinidos.
2. Calcula a média das duas notas e guarda numa outra variável “media”.
3. Mostra a média na terminal.