



# Power Up

# Introdução

# ao

# JavaScript



BYTES4FUTURE



FUNDAÇÃO AGA KHAN



comunidade de emoção



LISBOA  
2020 Capital Europeia  
da Cultura



ÁREA  
METROPOLITANA  
DE LISBOA



PRR  
Plano de Recuperação  
e Resiliência



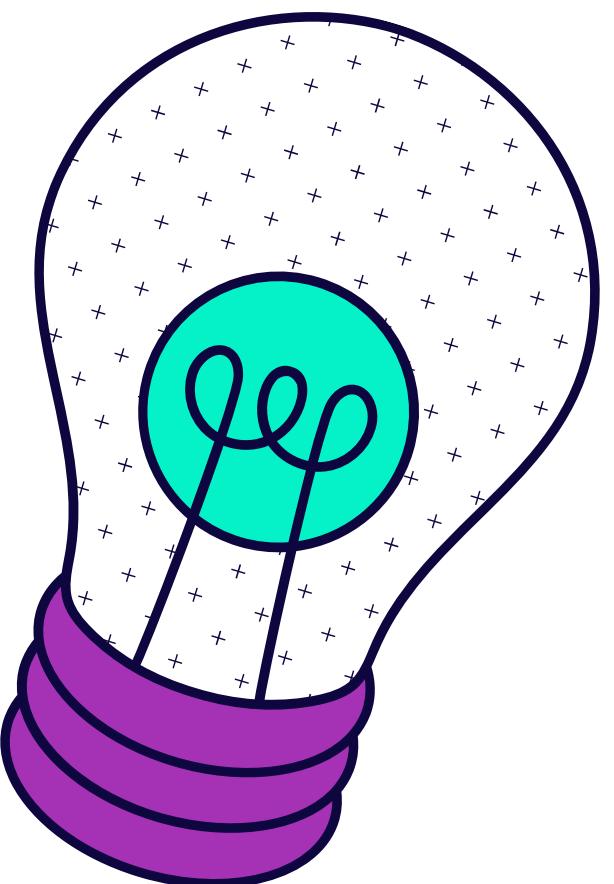
REPÚBLICA  
PORTUGUESA



Financiado pela  
União Europeia  
NextGenerationEU

search

# CONTEXTUALIZAÇÃO



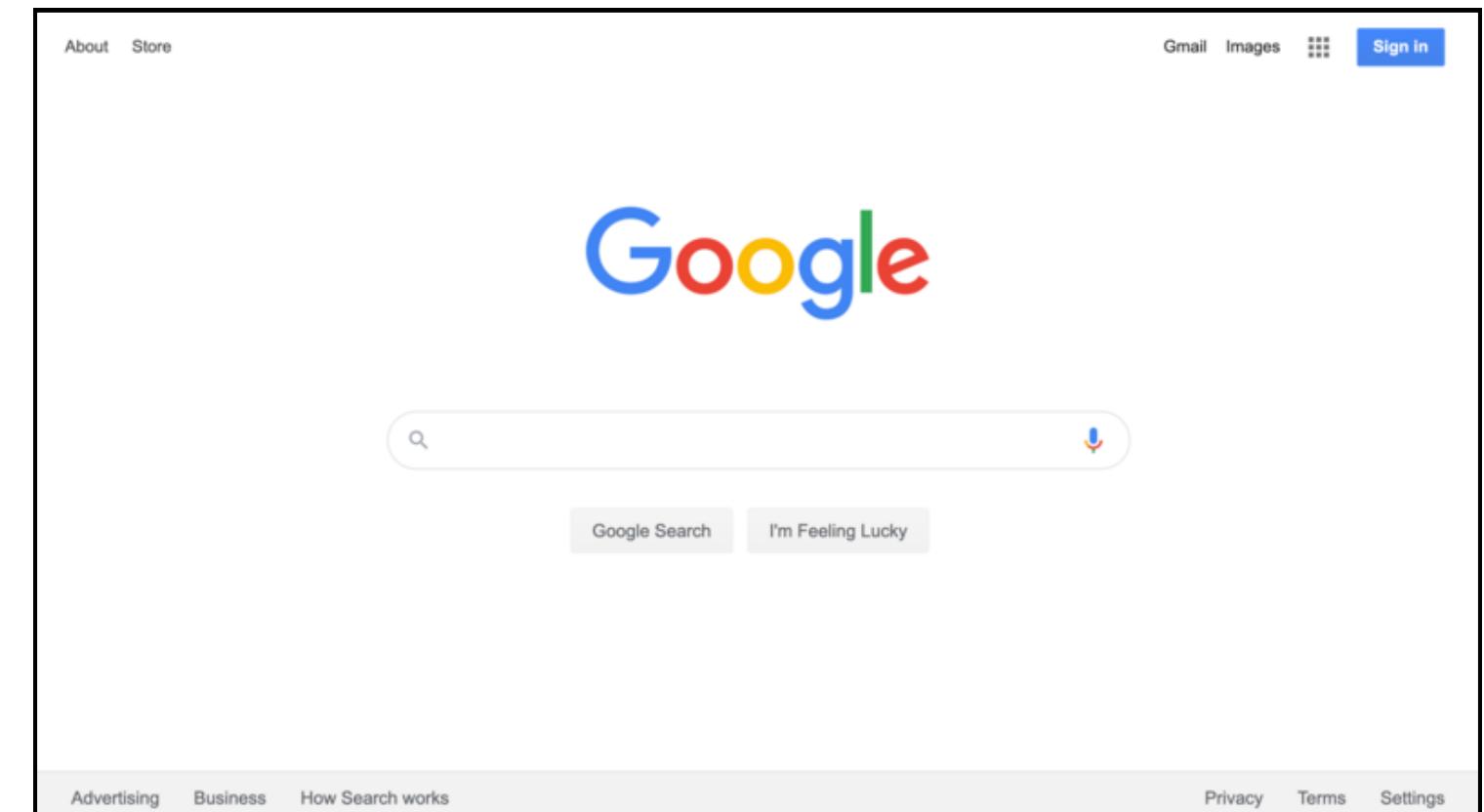
# História do Javascript

- 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!");
```



# Como criar um script

Há duas formas de adicionar Javascript ao ficheiro:

**É sempre colocado no final do body!**

Inline

```
<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>
<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

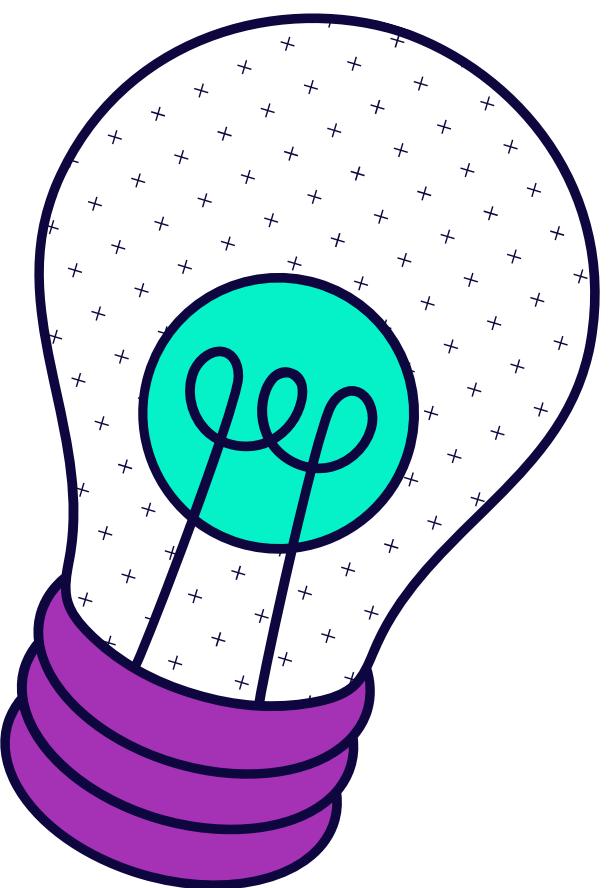
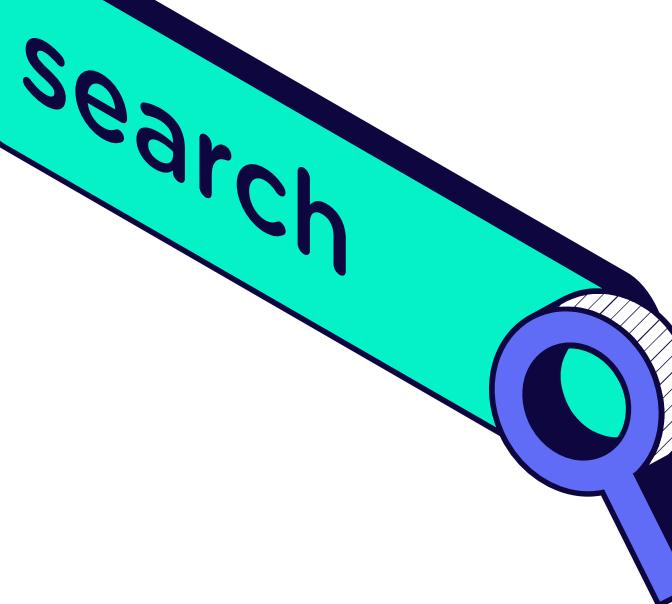
// Escreve o teu comentário aqui

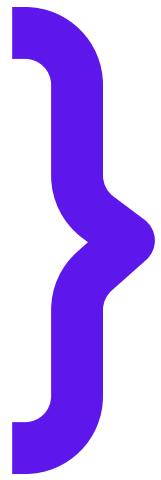
## Comentário de multiplas linhas:

/\* Escreve o teu  
comentário  
aqui \*/

}

# VÁRIAVEIS



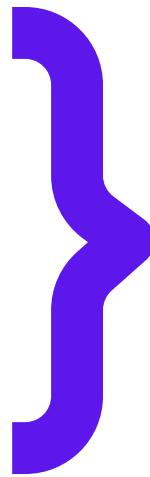


# Variáveis

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

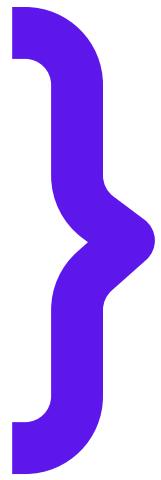
- Variáveis são nomes que guardam um valor que pode ser alterado.
- São declaradas com a palavra reservada `var` ou `let`. **Dá preferência ao `let`.**

```
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.

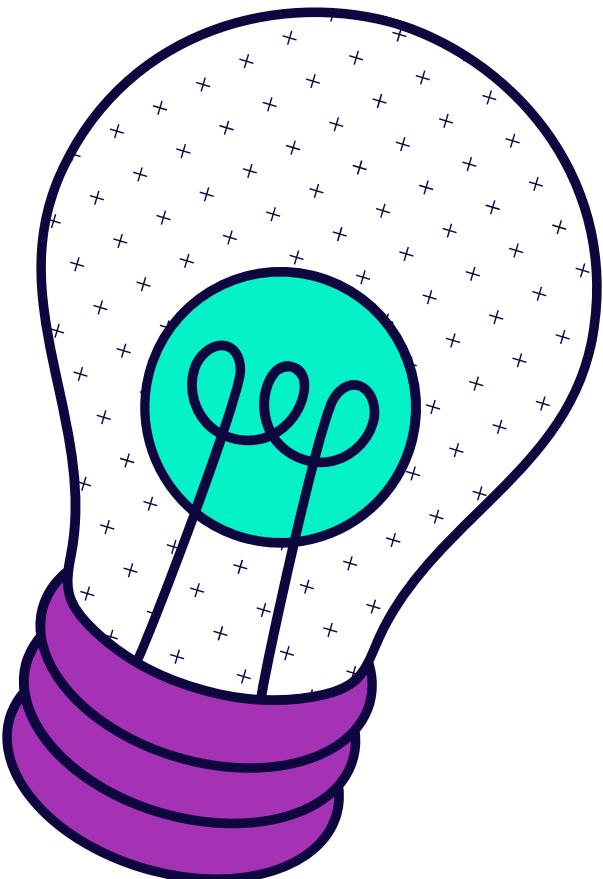
# Como Dar Nomes a Constantes

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

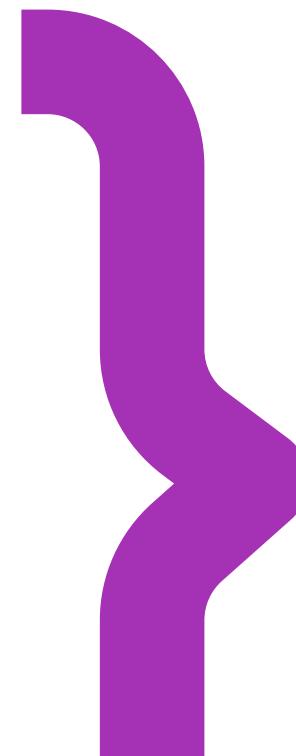
```
const MAX_NUMBER = 100
const MIN_NUMBER = 0
```

search

# QUIZ



# Vamos praticar





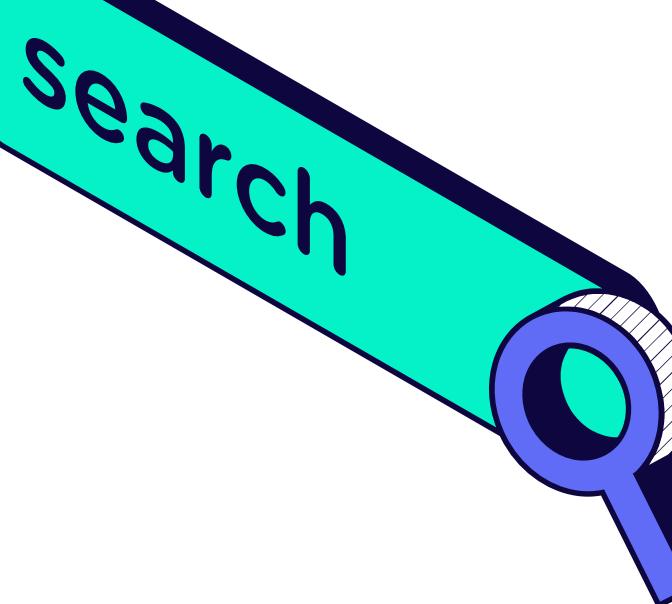
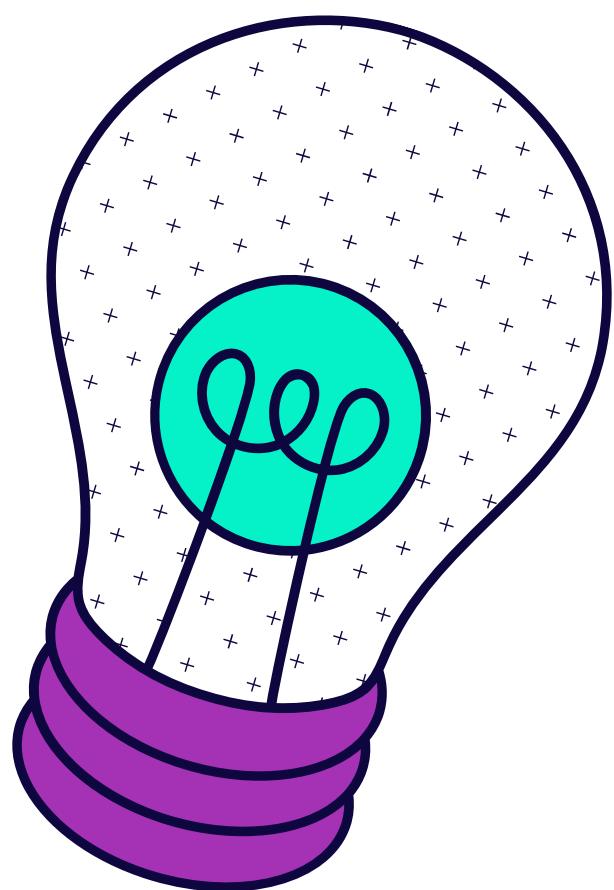
# Exercício:

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

**Imprime as variáveis na terminal.**

{}

# ESTRUTURA DE DADOS



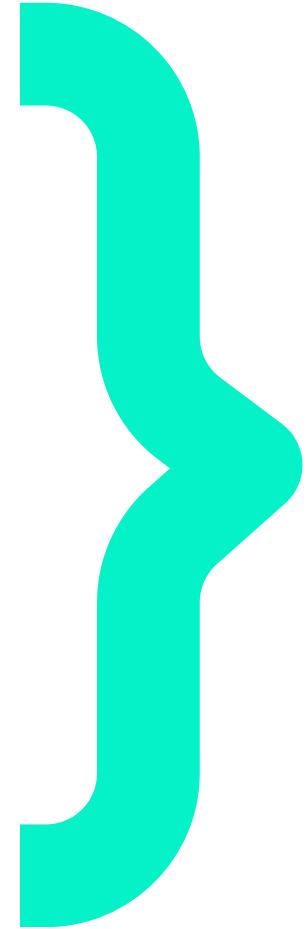
# Tipos de Dados

- 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.

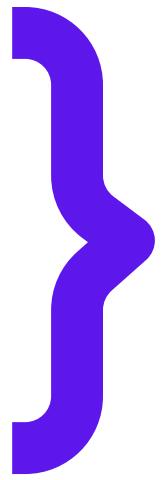
# Tipos de Dados Primitivos

Existem em JavaScript 6 tipos de dados primitivos:

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



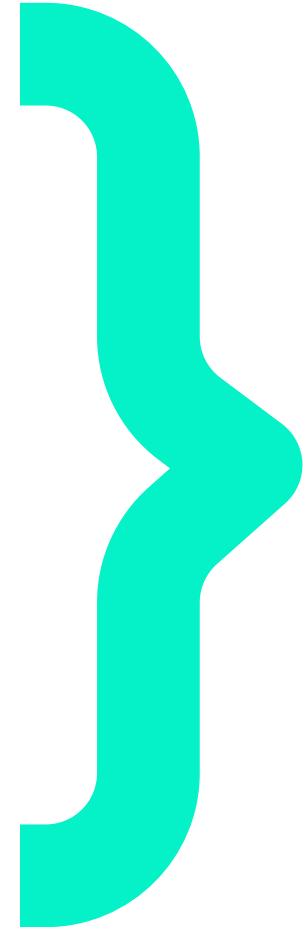
# Number



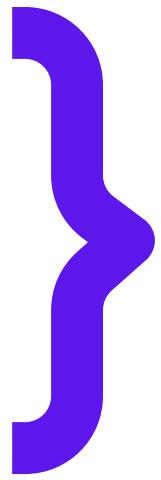
# Number

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

```
let n = 42
```



# **String**



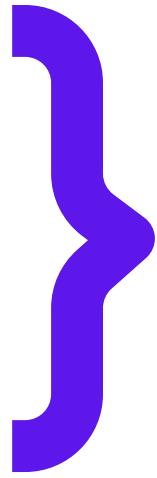
# String

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.

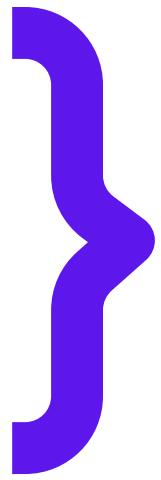
```
"Olá Mundo"  
'Olá Mundo'
```



# 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: \""

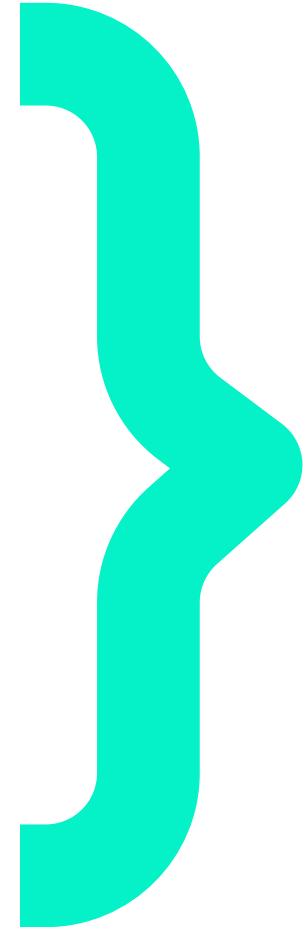
let barraInclinada = "Uma barra inclinada para a esquerda: \\";
console.log(barraInclinada);
// Resultado: "Uma barra inclinada para a esquerda: \\"
```

# 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.`;
```

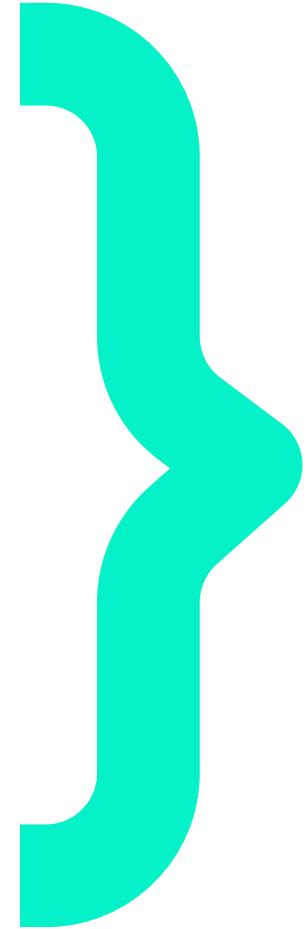


# Boolean

# Boolean

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

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



# Undefined

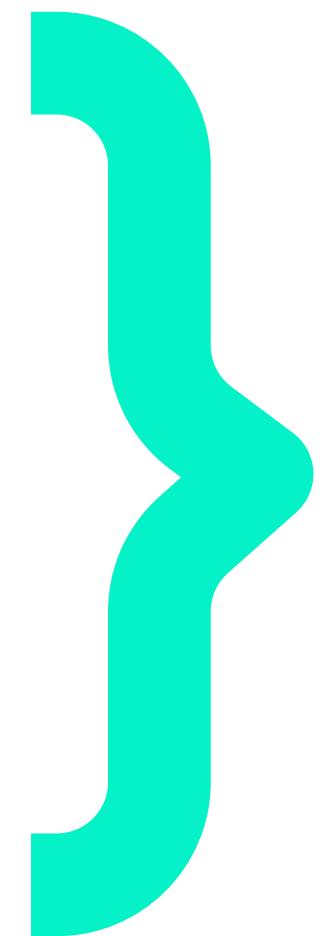
# Undefined

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

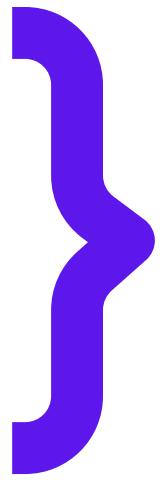
**Uma variável à qual ainda não foi atribuído um valor tem o valor undefined.**

Quando convertido para Boolean, é equivalente a false.

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



# BigInt



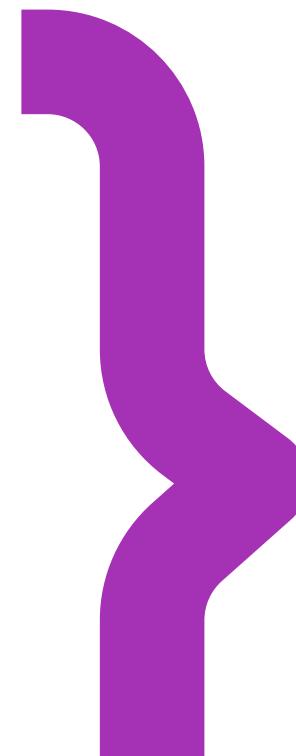
# BigInt

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



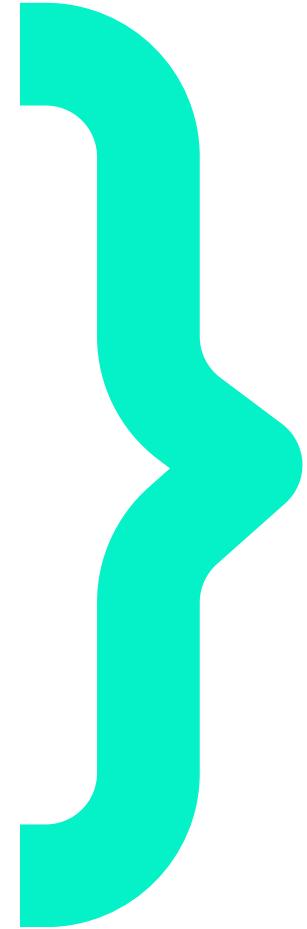
# Exercício:

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)

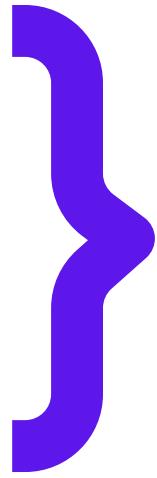
```
let contador = 5;  
contador++; // contador é 6
```

# Operadores de Comparaçāo

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

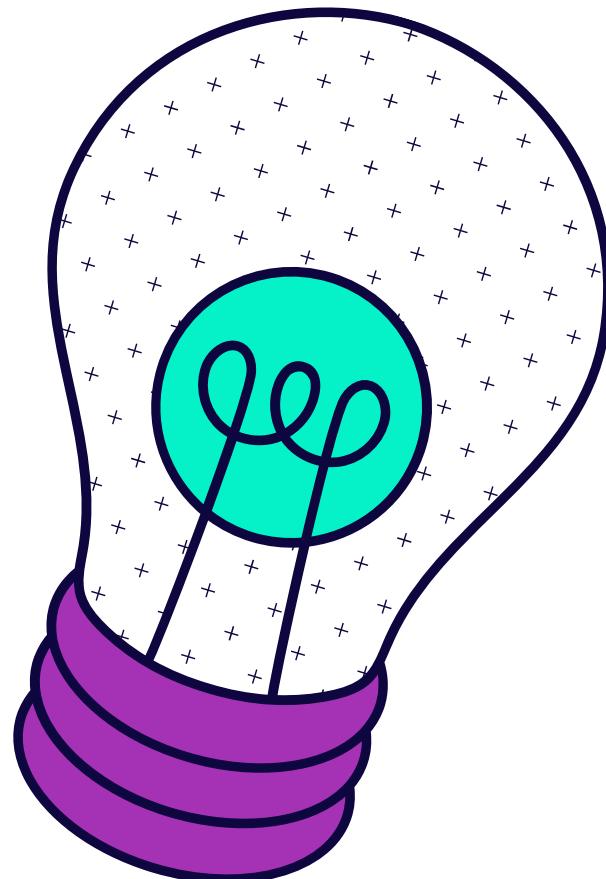
Usados para combinar **strings**.

- + (concatenação de strings)

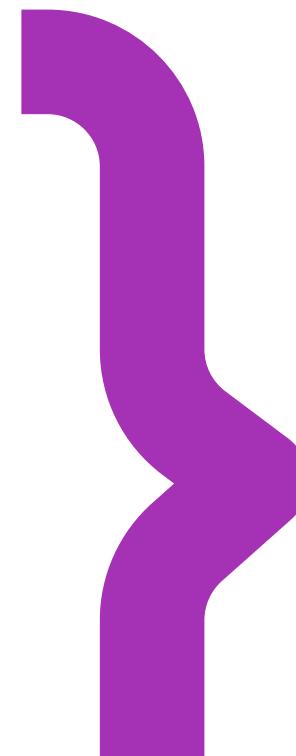
```
let nome = "Alice";
let sobrenome = "Smith";
let nomeCompleto = nome + " " + sobrenome; // "Alice Smith"
```

search

# 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.



# Power Up

# Introdução

# ao

# JavaScript



BYTES4FUTURE



FUNDAÇÃO AGA KHAN



comunidade de aprendizagem



LISBOA  
2020 Capital Europeia  
do Design



ÁREA  
METROPOLITANA  
DE LISBOA



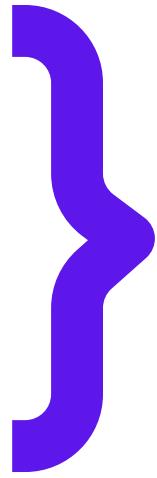
PRR  
Plano de Recuperação  
e Resiliência



REPÚBLICA  
PORTUGUESA



Financiado pela  
União Europeia  
NextGenerationEU



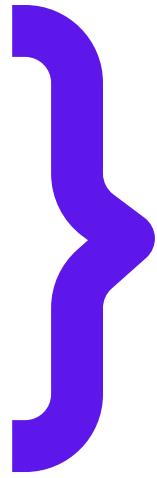
# Array

- Os arrays são usados para **armazenar múltiplos valores** e acessá-los por meio de índices.

## 1. Declarar um array:

Podes criar um array simplesmente utilizando parêntesis retos/colchetes [] e inserir os valores separados por vírgulas.

```
let frutas = ["maçã", "banana", "laranja"];
```

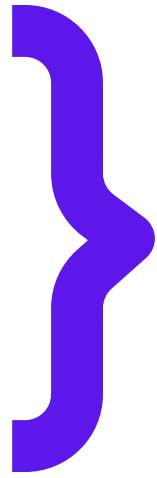


# Array

## 2. Aceder aos elementos de um array:

Os elementos de um array são acessados por **índices**, começando do índice 0 para o primeiro elemento.

```
let primeiraFruta = frutas[0]; // "maçã"
let segundaFruta = frutas[1]; // "banana"
```



# Array

## 3. Tamanho do array:

Podes obter o tamanho de um array utilizando a propriedade length.

```
let tamanho = frutas.length; // 3
```

## 4. Modificar elementos de um array:

Podes atribuir novos valores a elementos de um array usando o índice.

```
frutas[2] = "uva"; // Altera "laranja" para "uva"
```

# Array

## 5. Adicionar elementos a um array:

Podes adicionar elementos a um array usando o método push()

```
frutas.push("pera"); // Adiciona "pera" ao final do array
```

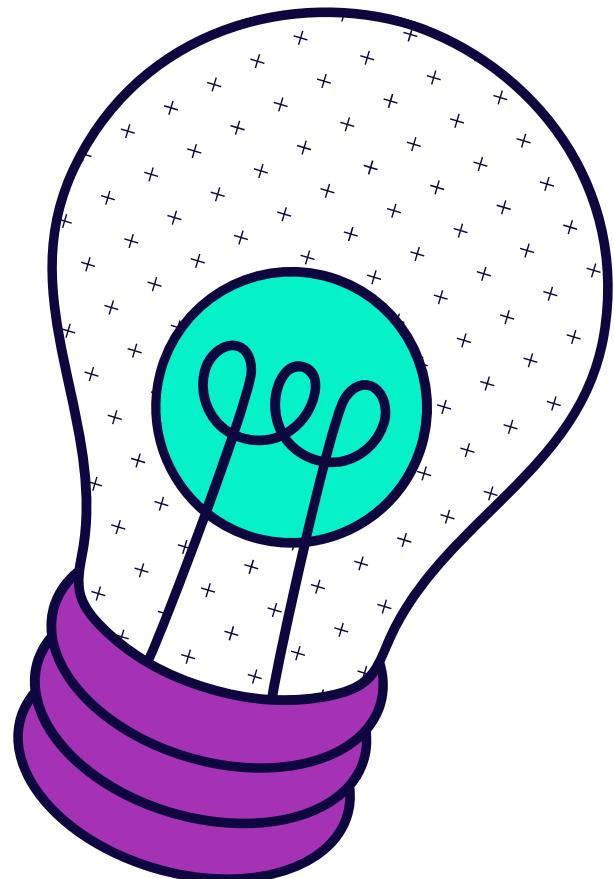
## 6. Remover elementos de um array:

Podes remover elementos de um array usando o método pop() para remover o último elemento.

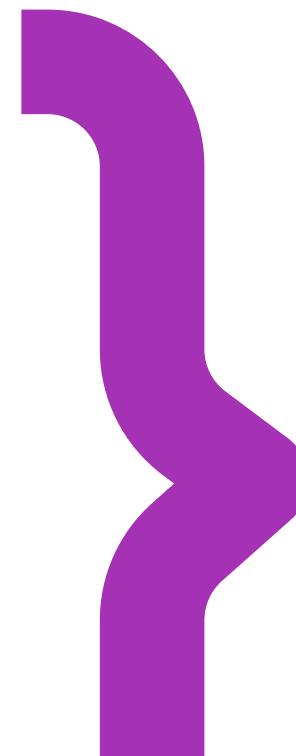
```
frutas.pop(); // Remove "pera" do final do array
```

search

# QUIZ



# Vamos praticar



# Exercício:

1. Cria um array chamado **frutas** com três nomes de frutas à tua escolha.
2. **Exibe** o conteúdo do array na terminal.
3. **Adiciona** uma nova fruta ao array.
4. Exibe o conteúdo do array novamente para verificar que a nova fruta foi adicionada.

# Exercício:

Considera um array inicial chamado “numeros” com os seguintes elementos:  
**1, 2, 3, 4 e 5.**

Utilizando apenas as operações de array **pop**, **push**, **shift**, **unshift** e manipulação de índices, realiza as seguintes tarefas:

1. Remove o último elemento do array.
2. Adiciona o número 6 ao final do array.
3. Remove o primeiro elemento do array.
4. Adiciona o número 0 no início do array.
5. Substitui o número no índice 2 por 10.
6. Imprime cada elemento do array numa linha.

# Exercício:

Considera um array inicial chamado “cores” com os seguintes elementos:  
"Vermelho", "Verde", "Azul" e "Amarelo".

Utilizando apenas as operações de array **pop**, **push**, **shift**, **unshift** e manipulação de índices, realize as seguintes tarefas:

1. Remove o último elemento do array.
2. Adiciona a cor "**Roxo**" ao final do array.
3. Remove o primeiro elemento do array.
4. Adiciona a cor "**Laranja**" no início do array.
5. Substitui a cor no índice 1 por "**Preto**".
6. Imprime cada elemento do array numa linha.

# Objetos

- Objetos permitem armazenar informações relacionadas em pares de **chave-valor**.
  - É uma maneira eficaz de representar entidades do mundo real com atributos e propriedades.

## 1. Declarar um objeto:

Neste exemplo, person é um **objeto com três propriedades: name, age, e job**. Cada propriedade tem um nome (a chave) e um valor associado.

```
const person = {  
  name: "Alice",  
  age: 30,  
  job: "teacher"  
};
```

# Objetos

## 2. Aceder às propriedades de um objeto:

Podes aceder às propriedades de um objeto usando a notação de ponto (dot notation) ou parênteses retos.

```
console.log(person.name); // Output: "Alice"
```

```
console.log(person["age"]); // Output: 30
```

# Objetos

## 3. Modificar Propriedades:

Podes modificar as propriedades de um objeto atribuindo novos valores.

```
person.age = 31;  
person.job = "developer";
```

## 4. AdicionarPropriedades:

Também é possível adicionar novas propriedades a um objeto a qualquer momento.

```
person.city = "New York";
```

# Objetos

## 5. Remover Propriedades:

Para remover uma propriedade de um objeto, podes usar o operador **delete**.

```
delete person.job;
```

{

# Objeto

Recapitulando...

```
// Criar um objeto person
const person = {
  name: "Alice",
  age: 30,
  job: "teacher"
};

// Modificar propriedades
person.age = 31;
person.job = "developer";

// Adicionar uma nova propriedade
person.city = "New York";

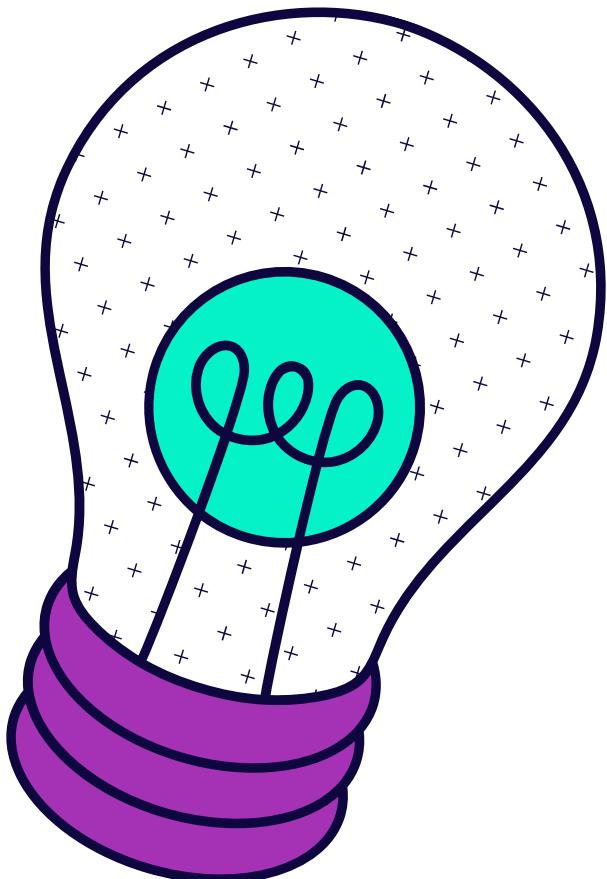
// Remover uma propriedade
delete person.job;
```

Objeto final:

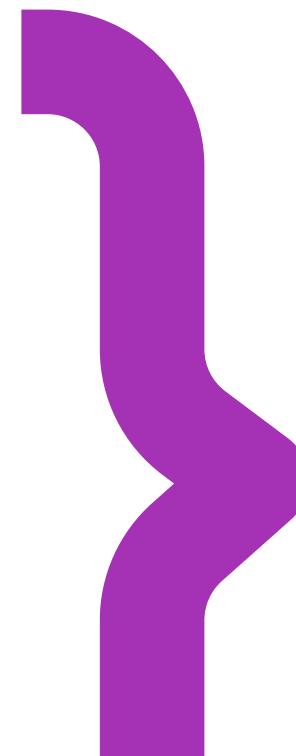
```
{
  name: "Alice",
  age: 31,
  city: "New York"
}
```

search

# QUIZ



# Vamos praticar



# Exercício:

1. Cria um objeto chamado **pessoa** com duas propriedades:
  - a. nome com o valor "João".
  - b. idade com o valor 25.
2. Exibe as informações da pessoa na terminal, incluindo o nome e a idade.

# Exercício:

1. Cria um objeto chamado **carro** com as seguintes propriedades:
  - a. **marca** (marca do carro).
  - b. **modelo** (modelo do carro).
  - c. **ano** (ano de fabrico do carro).
  - d. **cor** (cor do carro).
2. **Exibe** as informações do carro na terminal, incluindo a marca, o modelo, o ano e a cor.
3. **Altera** a cor do carro para "azul" e exibe novamente as informações atualizadas na terminal.
4. **Adiciona** uma nova propriedade chamada **quilometragem** com um valor à tua escolha.
5. **Exibe** todas as propriedades e os seus valores na terminal.



# Power Up

# Introdução

# ao

# JavaScript



BYTES4FUTURE



FUNDAÇÃO AGA KHAN



comunidade de prática



LISBOA  
Câmara Municipal



área metropolitana  
de lisboa



PRR  
projeto de recuperacão e reabilitaçao

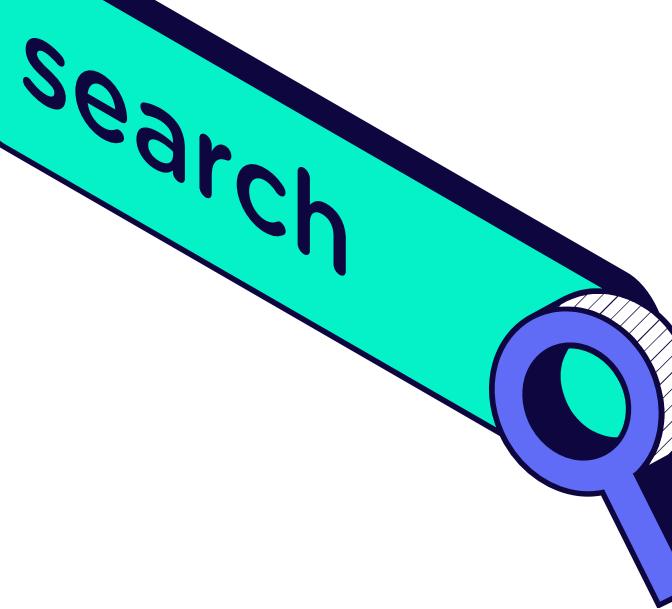
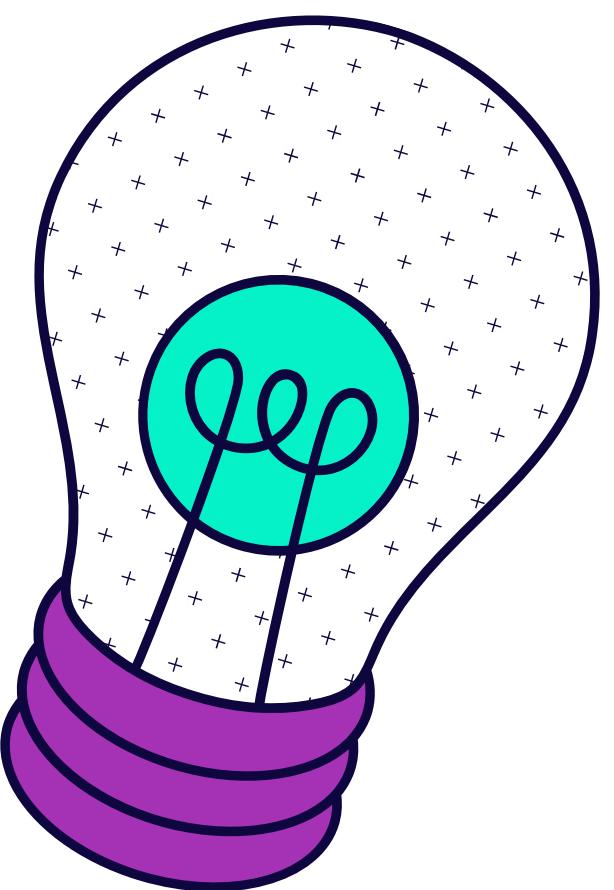
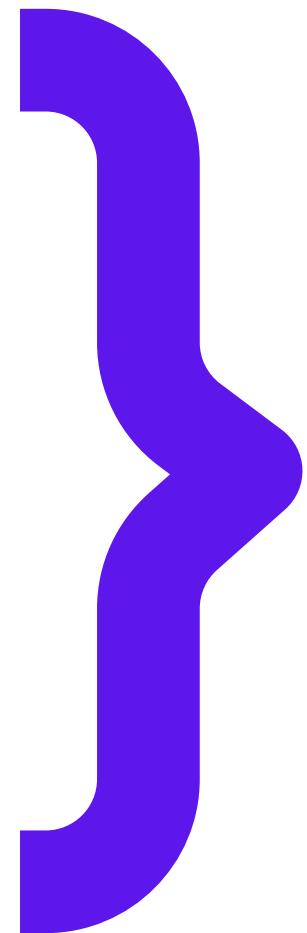


REPÚBLICA  
PORTUGUESA



Financiado pela  
União Europeia  
NextGenerationEU

# CONTROLO DE FLUXO

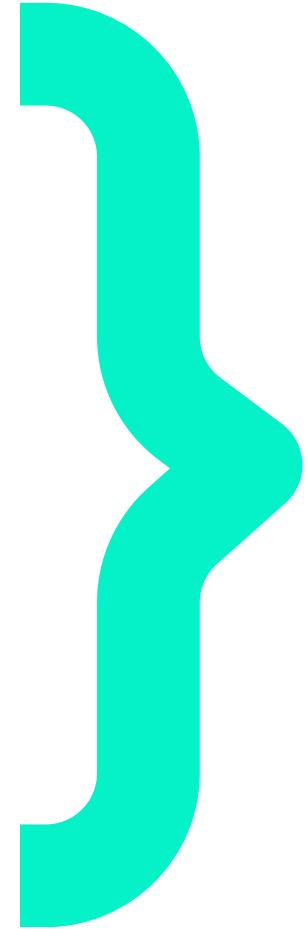


# Controlo de Fluxo

Com o que aprendeste até aqui consegue escrever um programa completamente sequencial e previsível, o que sinceramente torna-se muito aborrecido, muito rápido.

Existem duas formas principais de **controlar o fluxo** de um programa em JavaScript:

1. Executar partes do código apenas **se** uma determinada condição se verificar;
2. Executar partes do código **repetidamente, em ciclo**, até atingir a condição em que o programa sai do ciclo.



# **ESTRUTURAS CONDICIONAIS**

{

# if ... else

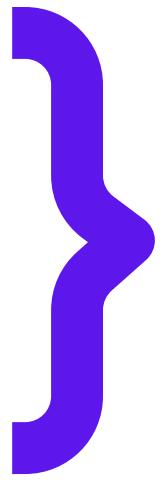
Uma das formas que permite controlar o fluxo de uma aplicação é utilizando o if, que pode ser opcionalmente seguido de um else.

A sintaxe de um if em JavaScript é:

```
if (condicao) {  
    // Código a ser executado se a condição for verdadeira  
} else {  
    // Código a ser executado se a condição for falsa  
}
```



If ... Else



# if ... else

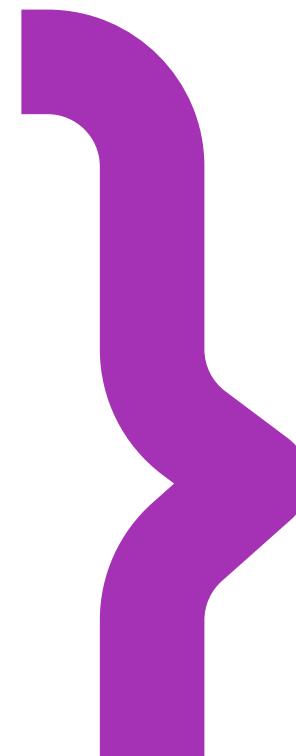
Um exemplo muito simples, se quiseres imprimir no terminal:

- par quando um número é par
- ímpar nos outros casos
- 7, se for o número 7
- Erro, se não for um número

console.log(...) permite-nos imprimir no terminal.

```
if (n === 7) {  
    console.log(n);  
} else if (n % 2 === 0) {  
    console.log('par');  
} else if (n % 2 === 1) {  
    console.log('ímpar');  
} else {  
    console.log('Erro')  
}
```

# Vamos praticar



# **Exercício:**

## **Verificar se é maior de idade:**

- Crie um código que verifica a idade de alguém, e imprime na terminal se a pessoa é maior de idade ou menor de idade.

# **Exercício:**

## **Verificação de Número Par ou Ímpar**

- Cria um programa que imprime na terminal, se um número é par ou ímpar.

# Exercício:

**Existem duas equipas de ginástica: Dolphins e Koalas. Elas competem entre si 3 vezes. A equipa vencedora com a pontuação média mais alta ganha um troféu!**

## As tuas tarefas:

1. Calcular a pontuação média para cada equipa, utilizando os dados de teste incluídos abaixo. A pontuação média dos Dolphins deve ser atribuída à variável `scoreDolphins`, e a pontuação média dos Koalas deve ser atribuída à variável `scoreKoalas`.
2. Comparar as pontuações médias das equipas para determinar o vencedor da competição e imprimir na terminal.
3. "*Os Dolphins ganham o troféu*" se os Dolphins vencerem, ou
4. "*Os Koalas ganham o troféu*" se os Koalas vencerem, ou
5. "*Ambos ganham o troféu*" se as pontuações médias forem iguais.

**DADOS DE TESTE:** Os **Dolphins** marcaram 96, 108 e 89. Os **Koalas** marcaram 88, 91 e 110.

# **Exercício:**

Escreve um código que atribua notas aos estudantes com base nas suas pontuações:

**90-100 -> A**

**70-89 -> B**

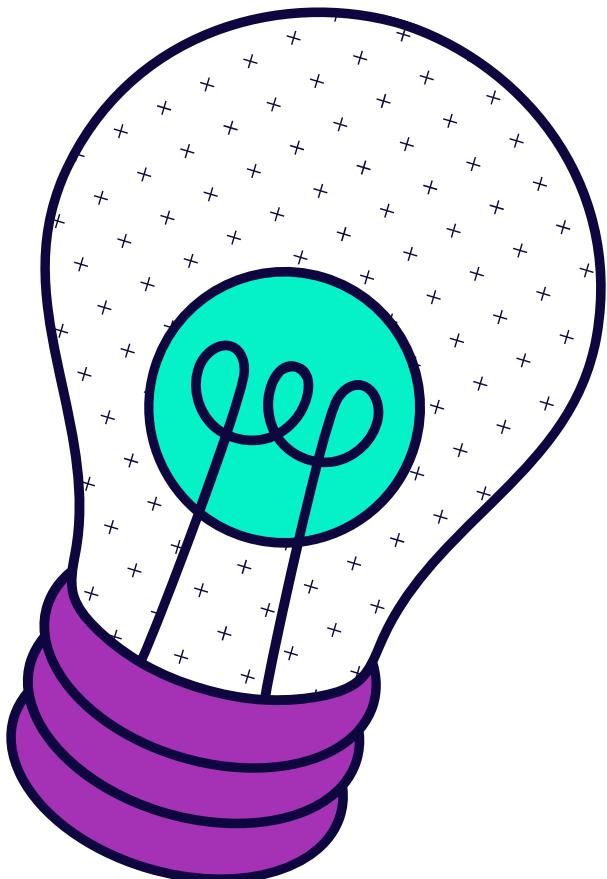
**60-69 -> C**

**50-59 -> D**

**0-49 -> F**

search

# QUIZ



# switch ... case

Nos casos em que existem muitas condições nas quais testas a mesma variável pode ser mais proveitoso utilizar um switch ... case

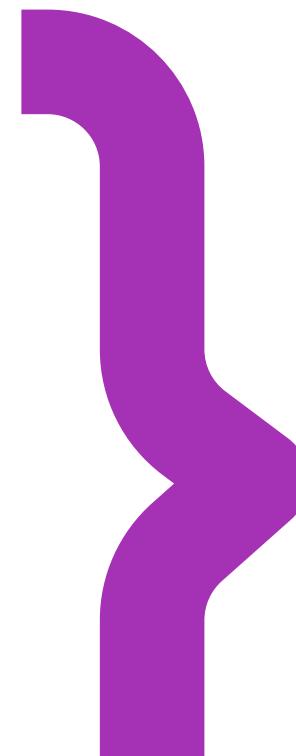
A sintaxe desta estrutura de controlo de fluxo é:

```
switch /* condição */ {  
    case valor_1:  
        // código a executar caso a variável tenha o valor_1  
        break;  
    case valor_2:  
        // código a executar caso a variável tenha o valor_2  
        break;  
    // ...  
    default:  
        // código a executar independentemente do valor  
        // excepto se houver breaks em todos os case  
}
```



Switch Case

# Vamos praticar



# Exercício:

- Implementa um código com que recebe um número que representa um mês do ano, e retorna o mês correspondente.

Exemplo:

```
mesesDoAno(1) // Janeiro  
mesesDoAno(10) // Outubro
```

# Exercício:

1. Crie um código para os dias da semana em que um número de 1 a 7 como argumento e retorna o nome do dia correspondente.

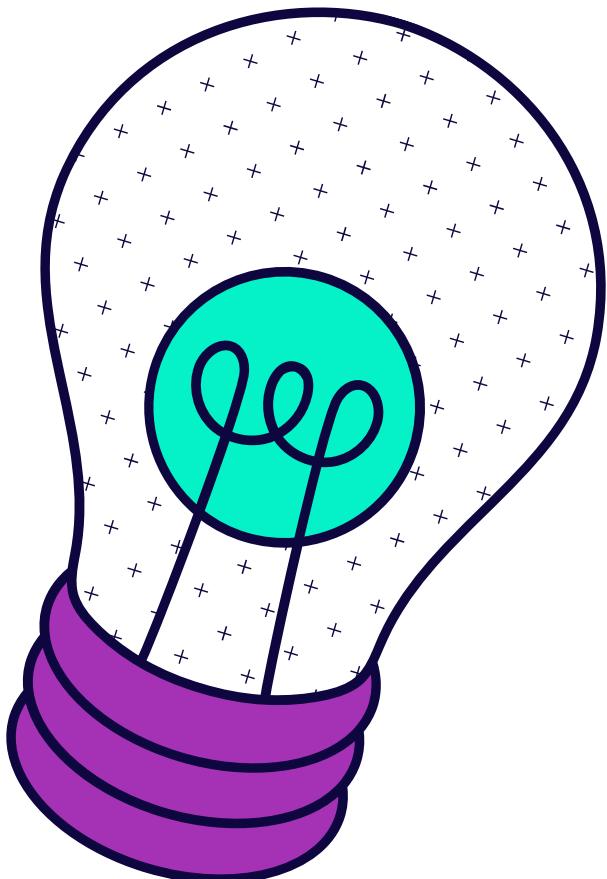
Utilize uma estrutura **switch-case** para implementar essa lógica.

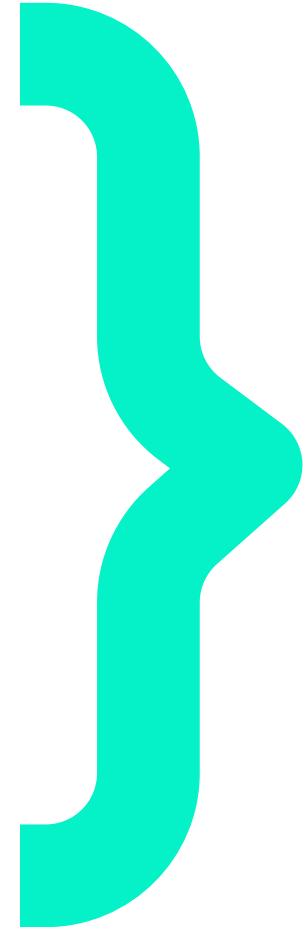
Exemplo:

```
console.log(diaDaSemana(1)); // Deve exibir "Domingo"  
console.log(diaDaSemana(3)); // Deve exibir "Terça-feira"  
console.log(diaDaSemana(8)); // Deve exibir "Dia inválido"
```

search

# QUIZ



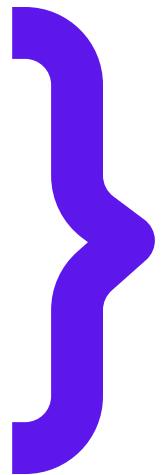


# CICLOS

# Ciclos

Muitos algoritmos exigem que uma determinada **instrução seja executada um determinado número de vezes**.

Os "**ciclos**" referem-se a estruturas de controlo que permitem a repetição de um conjunto de instruções várias vezes.



# while

O ciclo "**while**" é usado quando você deseja **repetir um conjunto** de instruções enquanto uma **condição específica for verdadeira**.

```
let i = 0;          // inicialização da variável
                  // utilizada para a condição
while (i < 10) { // Condição de continuação
    // Instruções a executar enquanto i for menor que 10
    i++;           // Incrementar i em 1
}
```



Ciclo WHILE

# }

# for

O ciclo "for" é usado quando você sabe exatamente quantas vezes deseja repetir um conjunto de instruções.

```
for (inicialização; condição; incremento) {  
    // Código a ser repetido enquanto a condição for verdadeira  
}
```

```
// Repara que todo o controlo do ciclo  
// está na primeira linha  
for (let i = 0; i < 10; i++) {  
    // Instruções a executar enquanto i for menor que 10  
}
```



Ciclo FOR

{

# for

**Enunciado: Tendo um array com números inteiros, descobre o valor da soma de todos os números.**

```
let arrayInteiros = [1, 39, 2392, 92, 163, 2, 62, 27, 29];
let soma = 0;

for( expressão           condição           incrementação
    let i = 0; i < arrayInteiros.length; i++ ) {

    soma += arrayInteiros[ i ];

}
```



Ciclo FOR

}

for

```
let arrayInteiros = [1, 39, 2392, 92, 163, 2, 62, 27, 29];
let soma = 0;

    // i < 9
for( let i = 0; i < arrayInteiros.length; i++ ) {

    0 += arrayInteiros[0 ];

}
```

}

for

```
let arrayInteiros = [1, 39, 2392, 92, 163, 2, 62, 27, 29];
let soma = 0;
```

```
for( let i = 0; i < arrayInteiros.length; i++ ) {
    // soma += arrayInteiros[i];
    0 += 1;
```

```
}
```

}

for

```
let arrayInteiros = [1, 39, 2392, 92, 163, 2, 62, 27, 29];
let soma = 0;

for( let i = 1; i < arrayInteiros.length; i++ ) {
    // soma += arrayInteiros[1];
    1 += 39;
}
```

}

for

```
let arrayInteiros = [1, 39, 2392, 92, 163, 2, 62, 27, 29];
let soma = 0;

for( let i = 2; i < arrayInteiros.length; i++ ) {
    // soma += arrayInteiros[2];
    40 += 2392;
}
```

}

for

```
let arrayInteiros = [1, 39, 2392, 92, 163, 2, 62, 27, 29];
let soma = 0;
```

```
for( let i = 3; i < arrayInteiros.length; i++ ) {
    // soma += arrayInteiros[3];
    2432 += 92;
}
```

}

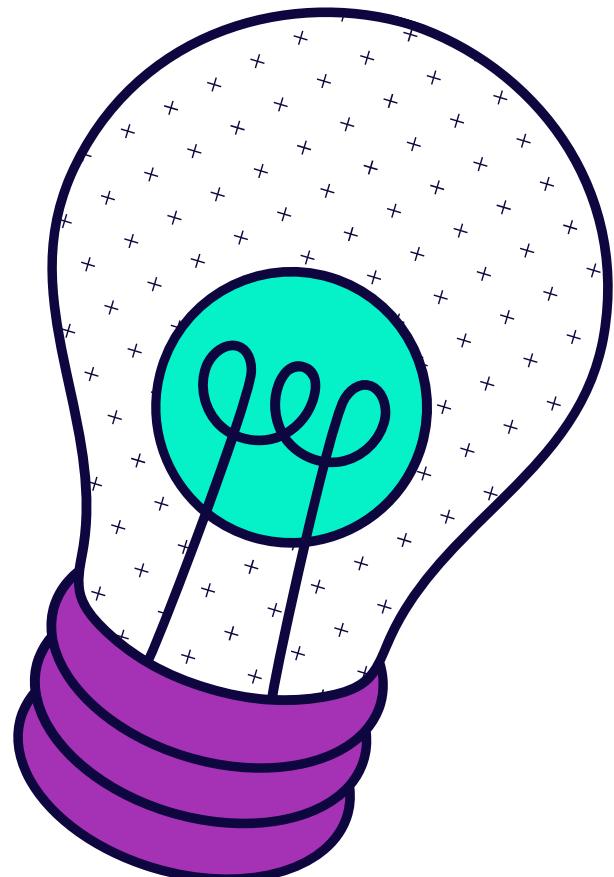
for

```
let arrayInteiros = [1, 39, 2392, 92, 163, 2, 62, 27, 29];
let soma = 0;

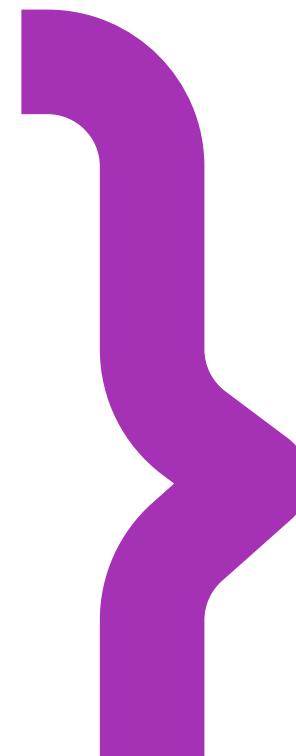
for( let i = 4; i < arrayInteiros.length; i++ ) {
    // soma += arrayInteiros[4];
    2524 += 163;
}
```

search

# QUIZ



# Vamos praticar



# Exercício:

1. Cria um código que recebe um número inteiro positivo e, em seguida, utiliza um **while loop** para contar de 1 até o número fornecido.
2. A função deve imprimir cada número no console à medida que conta.

Por exemplo, se a função for chamada com o argumento 5, a saída esperada no console seria:

```
1  
2  
3  
4  
5
```

# Exercício:

1. Repita o código que recebe um número inteiro positivo como argumento, só que utiliza um **loop for** para contar de 1 até o número fornecido.
2. Deve imprimir cada número no console à medida que conta.

Por exemplo, se a função for chamada com o argumento 5, a saída esperada no console seria:

```
1
2
3
4
5
```



# Power Up

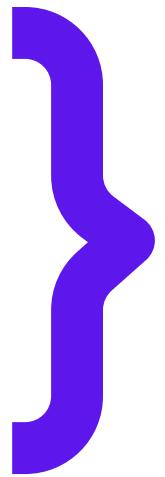
# Introdução

# ao

# JavaScript



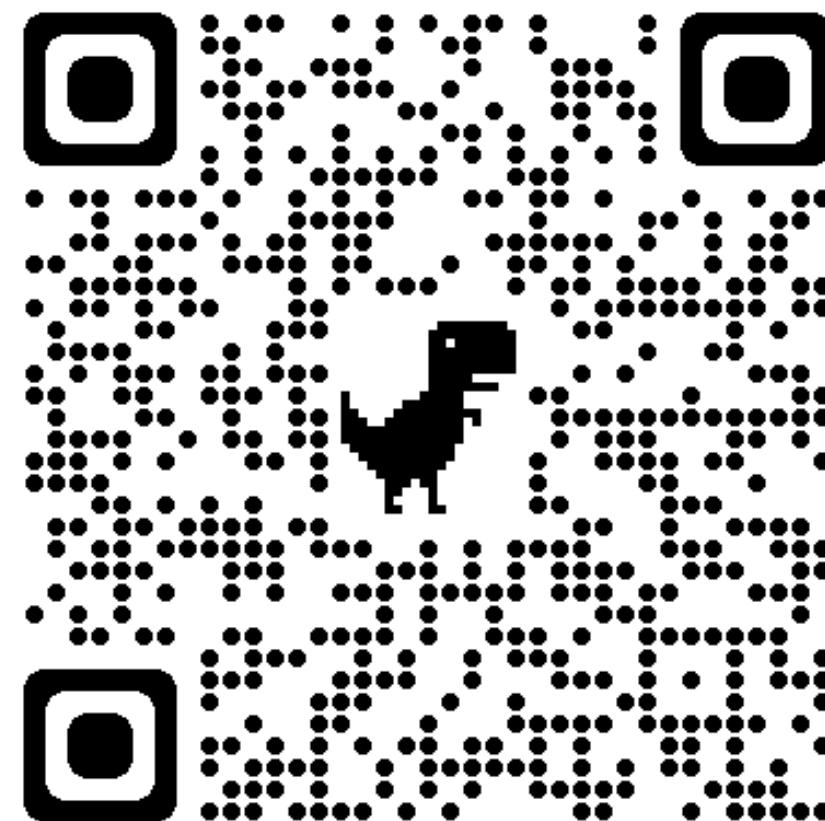
BYTES4FUTURE



# Funções

Funções em programação são **blocos de código** reutilizáveis.

Ou seja evitamos a repetição de um programa em diversas partes do código.



# Funções

## 1. Declaração de Função:

Uma função é definida com a palavra-chave **function**, seguida pelo nome da função e um conjunto de parâmetros, se houver.

O corpo da função é delimitado por chaves **{ }.**

```
function saudacao(nome) {  
    console.log(`Olá, ${nome}!`);  
}
```

# Funções

## 2. Chamada de Função:

Para executar o código dentro de uma função, você a chama pelo nome e fornece os argumentos necessários.

```
saudacao("Alice");
```

**Obs:** A função só é invocada quando ela é chamada.

# Funções

## 3. Parâmetros e Argumentos:

Os parâmetros são variáveis na definição da função, enquanto os argumentos são os valores reais fornecidos quando a função é chamada.

- **Parâmetros**: Variáveis listadas na definição da função.
- **Argumentos**: Valores reais passados para a função durante a chamada.

# Funções

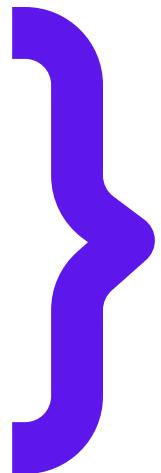
## 4. Retorno de Valores:

As funções podem retornar valores usando a palavra-chave **return**. O valor retornado pode ser atribuído a uma variável ou usado diretamente.

```
function soma(a, b) {  
    return a + b;  
}  
  
const resultado = soma(3, 5); // a é 3, b é 5. resultado é 8.
```

# Scope

- Scope define os limites onde pode ser usada uma determinada variável ou constante.
- É possível definir duas variáveis com o mesmo nome desde que seja em scopes diferentes.



# Scope

```
let globalVar = 10;

function exampleFunction() {
  console.log(globalVar); // A variável globalVar pode ser acessada dentro da função.
}
```

```
function exampleFunction() {
  let localVar = 5; // localVar tem escopo local à função exampleFunction.
  console.log(localVar);
}

// console.log(localVar); // Isso resultaria em um erro, pois localVar não é acessível aqui.
```

# Scope

Se existir um scope dentro de outro e existir uma colisão nos nomes das variáveis, é sempre usada variável mais interna.

```
let variavelExterna = "Externa";

function exemplo() {
    let variavelExterna = "Interna"; // Variável no escopo interno com o mesmo nome
    console.log("Variável interna:", variavelExterna); // Acessa a variável no escopo interno
}

exemplo(); // Chama a função

console.log("Variável externa:", variavelExterna); // Acessa a variável no escopo externo
```

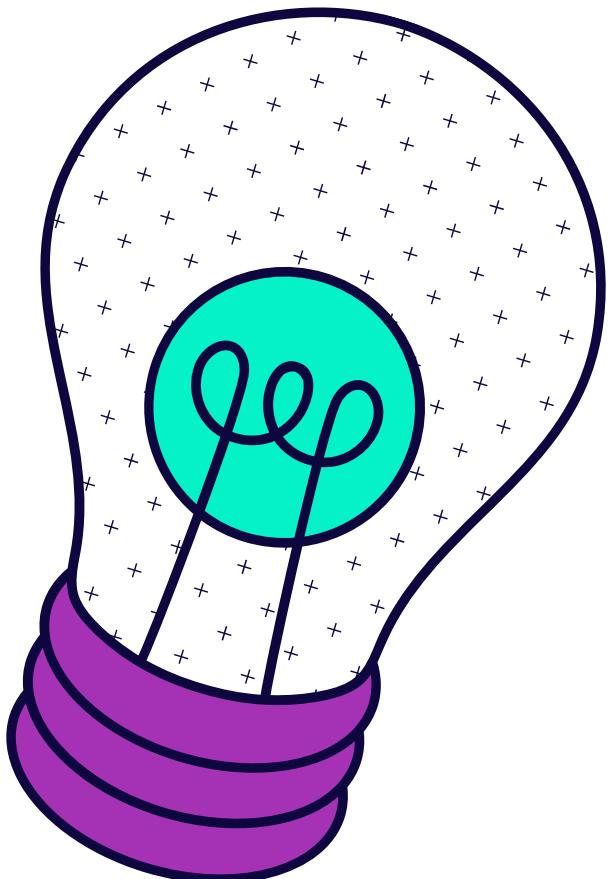
# Como Dar Nomes a Funções

- As funções costumam seguir a regra camelCase, ou seja, começam com minúsculas e a primeira letra de cada nova palavra é maiúscula.
- Normalmente os nomes das funções têm a particularidade de começar com um verbo.

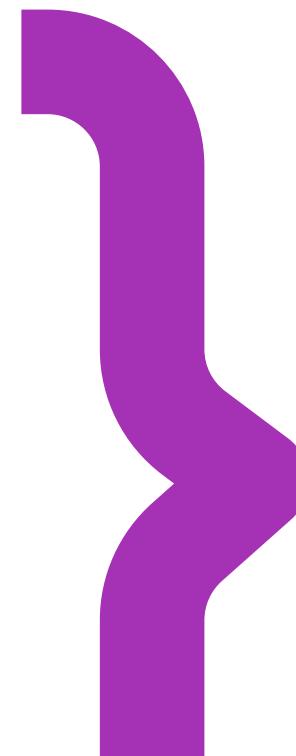
```
function computeBestScore() {  
    // ...  
}
```

search

# QUIZ



# Vamos praticar



# Exercício:

## Comprimento da String:

- Escreva uma função chamada **comprimentoString** que recebe uma string como parâmetro e retorna o comprimento da string.

# Exercício:

## Área do Retângulo:

- Implemente uma função chamada **calcularAreaRetangulo** que recebe a largura e a altura de um retângulo como parâmetros e retorna a área.

# Exercício:

## Concatenar Strings:

- Escreva uma função chamada **concatenarStrings** que recebe duas strings como parâmetros e retorna uma única string resultante da concatenação das duas, com um espaço entre elas.

# Exercício:

## Somar

- Escreva uma função chamada **somar** que recebe duas variáveis a e b como parâmetros e retorna valor numérico da soma das duas variáveis, guarde o resultado numa variável chamada soma.

# Exercício:

## Dobro e Triplo:

- Crie uma função chamada **dobroETriplo** que recebe um número como parâmetro e retorna um array contendo o dobro e o triplo desse número.

# Exercício:

## Raiz Quadrada:

- Escreva uma função chamada **calcularRaizQuadrada** que recebe um número como parâmetro e retorna a raiz quadrada desse número.

# Exercício:

## Calcular Idade em Dias:

- Crie uma função chamada **calcularIdadeEmDias** que recebe a idade de uma pessoa como parâmetro e retorna a idade em dias. Considere que um ano tem 365 dias.

# Exercício:

## Criar Objeto de Pessoa:

- Crie uma função chamada **criarPessoa** que recebe parâmetros como nome, idade e cidade, e retorna um objeto representando uma pessoa.

# Exercício:

## Adicionar Propriedade:

- Escreva uma função chamada **adicionarPropriedade** que recebe um objeto e adiciona uma nova propriedade a ele.

# Exercício:

## Remover Propriedade:

Escreva uma função chamada **removerPropriedade** que recebe um objeto e o nome de uma propriedade, retornando um novo objeto sem a propriedade especificada.