

INICIANDO JAVASCRIPT



**QUE A FORÇA ESTEJA COM VOCÊ.
PROF. FERNANDO LUCAS**

LUCAS FERNANDO PROF.

INTRODUÇÃO AO JAVASCRIPT

JAVASCRIPT É UMA LINGUAGEM DE PROGRAMAÇÃO USADA PARA ADICIONAR LÓGICA, COMPORTAMENTO E INTERATIVIDADE.

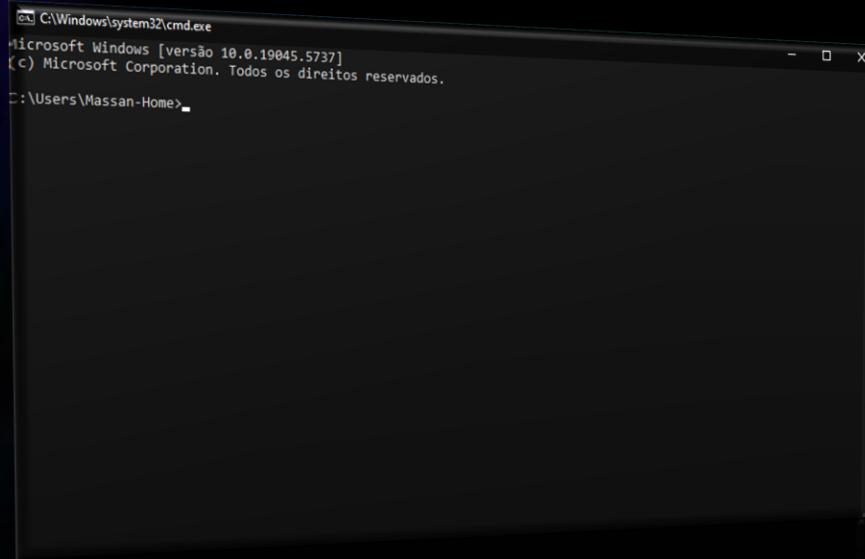
ELA É LEVE, INTERPRETADA, MULTIPARADIGMA E MUITO USADA TANTO NO NAVEGADOR QUANTO NO BACKEND (COM NODE.JS).



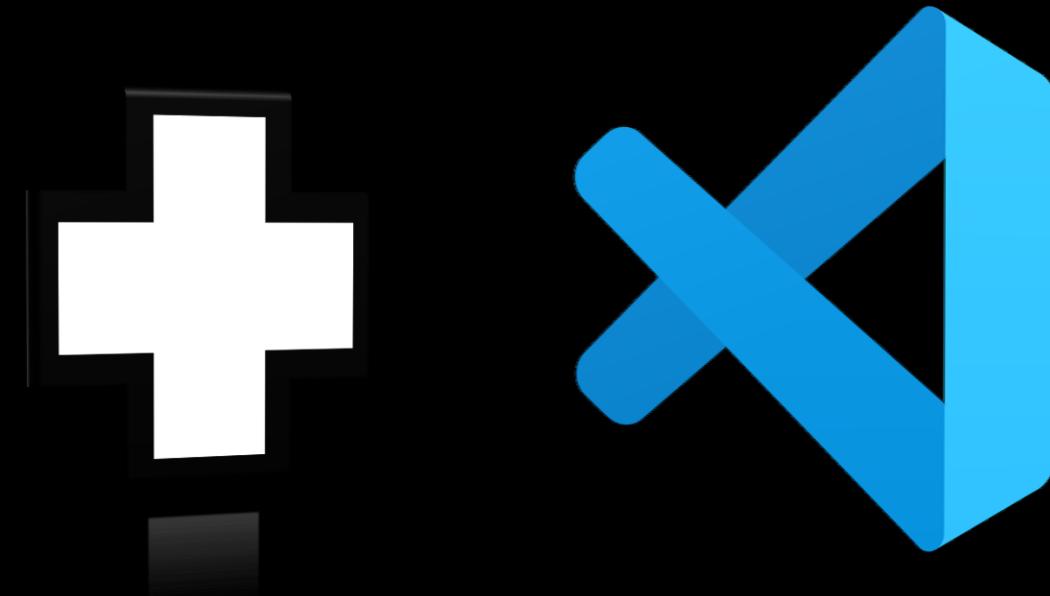
 **GLOSSÁRIO:**
MULTIPARADIGMA
BACKEND



VAMOS UTILIZAR O CMD PARA CRIAR UMA NOVA PASTA E ABRIR A MESMA NO VS CODE.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [versão 10.0.19045.5737]
(c) Microsoft Corporation. Todos os direitos reservados.
C:\Users\Massan-Home>
```



COMANDOS –

DIR – LISTA OS ARQUIVOS

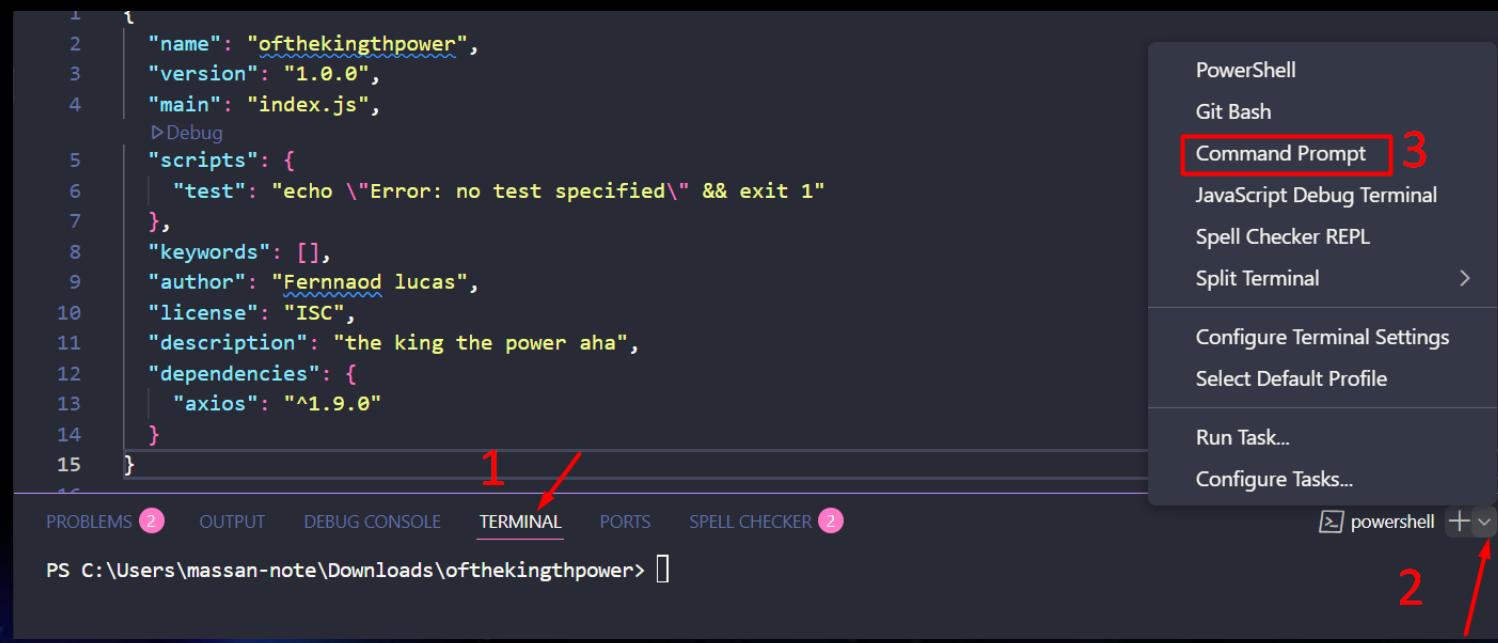
CD 'NOME DA PASTA' – NAVEGA NAS PASTAS

MKDIR – CRIA UMA PASTA NOVA

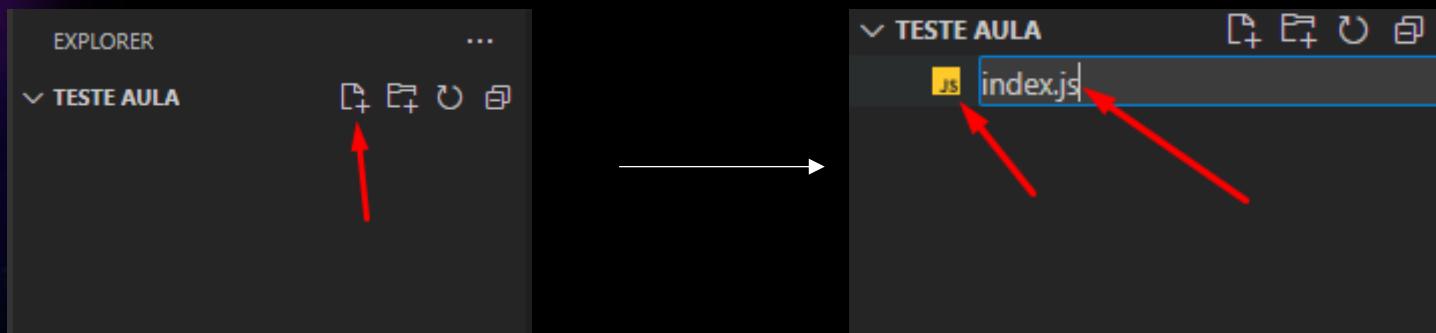
CODE . – ABRE O VS CODE NA PASTA QUE VOCÊ ESTA.

SEMPRE UTILIZAMOS O CMD PARA EXECUTAR NOSSOS PROJETOS. VAMOS CONTINUAR UTILIZANDO ELE MAS AGORA DENTRO DO VS CODE.

NO VS CODE APERTE **CTRL + J PARA ABRIR O TERMINAL.
CASO ELE VENHA COM O **POWERSHELL** VOCÊ DEVE
ALTERAR PARA O **COMMAND PROMPT** (QUE É O **CMD**).**



criando arquivos em javascript no vscode.



CRIE UM NOVO ARQUIVO **JS.**

ESCOLHA UM NOME QUE POR PADRÃO É INDEX, E REPARE QUE O QUE VOCÊ COLOCA DEPOIS DO '.' É QUE DEFINE O TIPO DO ARQUIVO.

PODERIA SER:

INDEX.HTML

INDEX.CSS

INDEX.JS

INDEX.PKG

ENTRE OUTRAS 1000 LINGUAGENS QUE EXISTEM.

CRIANDO UMA CONSTANTE E MOSTRANDO ELA NO CONSOLE.

```
JS teste.js > ...
1 const menssage = "Hello World"
2 console.log(menssage)
```

**COPIE NO SEU NOVO
ARQUIVO.JS**

CTRL J

PARA ABRIR O CONSOLE

**VERIFIQUE SE VOCÊ ESTA
NA PASTA CORRETA.**



**AGORA NO CONSOLE ESCREVA.
O NOME DO MOTOR (espaço) O NOME DO ARQUIVO QUE VOCÊ QUER
EXECUTAR.**

```
JS teste.js
```

VARIÁVEIS E TIPOS DE DADOS

VARIÁVEIS SÃO ESPAÇOS NA MEMÓRIA PARA GUARDAR DADOS.

EXEMPLOS:

LET

= VALOR QUE PODE MUDAR.

CONST

= VALOR CONSTANTE (NÃO PODE SER ALTERADO DURANTE A EXECUÇÃO DO CODIGO).

VAR

= FORMA ANTIGA (EVITAR)

IMAGINE AS VARIÁVEIS COMO CAIXAS PARA GUARDAR DADOS!

JÁ OS TIPOS DE DADOS NÃO O QUE A VARIÁVEL GUARDA E SIM COMO SE DEFINE O QUE ELA ESTA GUARDANDO.

EXEMPLO:

```
let exemplo1 = "1"
```

**DECLARANDO QUE
UM DADO SERÁ
GUARDADO E QUE
ELE PODE SER
ALTERADO.**

**DECLARANDO O
NOME DA VARIÁVEL
PARA QUE
POSSAMOS USA-LA
DEPOIS. (ESSE NOME
VOCÊ QUE
ESCOLHE)**

**O TIPO DO DADO QUE ESTA
SENDO GUARDADO. NESSE
CASO UMA STRING OU SEJA
UM TEXTO.**

```
let exemplo2 = 1
```

**O TIPO DO DADO QUE ESTA
SENDO GUARDADO. NESSE
CASO UM INT OU SEJA UM
VALOR INTEIRO.**

E QUAL A DIFERENÇA?

EXEMPLO DE DIFERENÇA

EXERCICIO:

Crie um arquivo chamado exec1.js
copie o seguinte código nele:

```
let exemplo1 = "1"  
let exemplo2 = 1  
console.log(exemplo1+exemplo1)
```

Agora execute o código!

O NOME DO MOTOR (espaço) **O NOME DO ARQUIVO QUE VOCÊ QUER EXECUTAR.**
O QUE ACONTECEU?

AGORA EXECUTE DESSA OUTRA FORMA:

```
console.log(exemplo2+exemplo2)
```

O QUE ACONTECEU?

**VOCÊ NOTOU QUE DE ACORDO COM O TIPO DO DADO QUE A VARAIVEL GUARDA
O RESULTADO PODE SER ALTERADO.**

AGORA VAMOS FALAR DOS:

TIPOS PRINCIPAIS



TIPOS DE DADOS

STRING

= **TEXTO** – **EXEMPLO:** let exemplo1 = "1"

NUMBER

= **VALOR INTEIRO OU DECIMAL** – **EXEMPLO:** let exemplo2 = 1

BOOLEAN

= **VERDADEIRO OU FALSO** – **EXEMPLO** - let exemplo3 = true
let exemplo4 = false

NULL

= **NULO, VAZIO PROPOSITALMENTE** – **EXEMPLO** - let exemplo5 = null

UNDEFINED

= **NÃO DEFINIDO** – **EXEMPLO** - let exemplo6 = undefined

OBJECT

ARRAY

FUNCTION

- **VEREMOS POSTERIAMENTE**



TEMPLATE STRINGS

JÁ ENTENDEMOS, QUE OS DADOS QUE ESTÃO ENTRE ASPAS SÃO STRINGS.

EXEMPLO: ' **ISTO É UMA FRASE E NÃO REPRESENTA UM COMANDO.** '

**QUER DIZER QUE NÃO PODEMOS CHAMAR COMANDOS OU VARIÁVEIS DENTRO
DESSA STRING OU FRASE.**

VAMOS IMAGINAR QUE TEMOS A SEGUINTE SITUAÇÃO.

TEMOS ESSA A SEGUINTE VARIÁVEL:

```
let notaDoAluno1 = 7.4
```

**IMAGINA QUE QUEREMOS UTILIZAR O VALOR DESSA VARIÁVEL DENTRO
DE UMA FRASE.**

MAS ANTES VOCÊ CONSEGUE RESPONDER ESSAS DUAS QUESTÕES.

QUAL O VALOR QUE ESSA VARIÁVEL notaDoAluno1 **ESTA GUARDANDO?**

QUAL O TIPO DELA NO MOMENTO ATUAL?



TEMPLATESTRING

ISSO MESMO O VALOR QUE ELA GUARDA É **7.4**
E O TIPO DELA É **NUMBER**.

AGORA CONTINUEMOS A SEGUINTE SITUAÇÃO.

QUEREMOS ESCREVER O VALOR DESSA VARIÁVEL DIRETO NA FRASE.

```
let notaDoAluno1 = 7.4  
console.log("a nota do aluno foi notaDoAluno1")
```

EXECUTE O CÓDIGO VERIFIQUE O QUE ACONTECEU



TEMPLATESTRING

NOTE QUE FALHAMOS MISERAVELMENTE.

TENTE ASSIM AGORA:

TROQUE AS ASPAS ""

POR GRAVE ``

```
console.log(`a nota do aluno foi notaDoAluno1`)
```

EXECUTE O CÓDIGO E VEJA MAGICA

AGORA PRECISAMOS ENTENDER COMO FUNCIONA A MATEMÁTICA NO CÓDIGO.

AGORA VAMOS FALAR DE:

OPERADORES E EXPRESSÕES



OPERADORES

OPERADORES SÃO SÍMBOLOS QUE REALIZAM OPERAÇÕES COM VARIÁVEIS E VALORES.

◆ PRINCIPAIS TIPOS DE OPERADORES EM JAVASCRIPT:

1. Operadores Aritméticos

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão
**	Exponenciação



OPERADORES

EXEMPLO 1: QUESTÃO SIMPLES (VISUAL)

```
console.log(5 + 3); // 8  
console.log(10 % 3); // 1
```



OPERADORES

EXEMPLO 2: USO PRÁTICO COMUM

```
let preco = 20;  
let quantidade = 3;  
let total = preco * quantidade;  
console.log(`Total a pagar: R$, ${total}`);
```



OPERADORES



EXERCÍCIO

**CRIE DUAS VARIÁVEIS: NOTA1 E NOTA2.
CALCULE A MÉDIA DAS NOTAS E MOSTRE NO CONSOLE:
"A MÉDIA DO ALUNO FOI: X"**



OPERADORES

OPERADORES DE ATRIBUIÇÃO

◆ OPERADORES DE ATRIBUIÇÃO EM JAVASCRIPT:

Operador	Exemplo	Equivalente
=	x = 5	atribui 5 a x
+=	x += 3	x = x + 3
-=	x -= 2	x = x - 2
*=	x *= 4	x = x * 4
/=	x /= 4	x = x / 4



OPERADORES

EXEMPLO 1: QUESTÃO SIMPLES (VISUAL)

```
let a = 10;  
a += 5;  
console.log(a);
```



OPERADORES

EXEMPLO 2: USO PRÁTICO COMUM

```
let saldo = 100;  
saldo -= 20;  
console.log("Saldo atual:", saldo);
```



OPERADORES



EXERCÍCIO

**CRIE UMA VARIÁVEL ECONOMIAS COM VALOR 0.
DEPOIS, ADICIONE 50 DUAS VEZES E SUBTRAIA 30.
MOSTRE O VALOR FINAL.**



OPERADORES

OPERADORES DE COMPARAÇÃO

◆ OPERADORES DE COMPARAÇÃO EM JAVASCRIPT:

Operador	Significado
<code>==</code>	Igual (com conversão)
<code>===</code>	Igual (valor e tipo)
<code>!=</code>	Diferente (com conversão)
<code>!==</code>	Diferente (valor e tipo)
<code>></code>	Maior que
<code><</code>	Menor que
<code>>=</code>	Maior ou igual
<code><=</code>	Menor ou igual
<code><=</code>	Menor ou igual



OPERADORES

EXEMPLO 1: QUESTÃO SIMPLES (VISUAL)

```
console.log(5 == "5");
console.log(5 === "5");
```



OPERADORES

EXEMPLO 2: USO PRÁTICO COMUM

```
let idade = 18;  
console.log(idade >= 18);
```



OPERADORES

EXEMPLO 3: USO PRÁTICO EM QUESTÕES

```
let senhaCorreta = "1234";
let senhaDigitada = "1234";
console.log(senhaCorreta === senhaDigitada);
```



OPERADORES



EXERCÍCIO

**CRIE UMA VARIÁVEL TEMPERATURA COM UM VALOR.
DEPOIS VERIFIQUE SE A TEMPERATURA ESTÁ ACIMA DE 30 GRAUS.
MOSTRE ESTÁ QUENTE? TRUE/FALSE.**

OPERADORES

OPERADORES DE COMPARAÇÃO

◆ OPERADORES LÓGICOS:

Operador	Significado	Exemplo
<code>&&</code>	E (AND)	<code>true && false</code> → <code>false</code>
<code> </code>	OU (OR)	<code>true false</code> → <code>true</code>
<code>!</code>	NÃO (NOT)	<code>!true</code> → <code>false</code>



OPERADORES

EXEMPLO 1: QUESTÃO SIMPLES (VISUAL)

```
console.log(true && false);
```

```
console.log(true || false);
```



OPERADORES

EXEMPLO 2: USO PRÁTICO COMUM

```
let idade = 20;  
let temCarteira = true;  
let podeDirigir = idade >= 18 && temCarteira;  
console.log("Pode dirigir?", podeDirigir);
```



OPERADORES

EXEMPLO 3: USO PRÁTICO EM QUESTÕES

```
let logado = true;  
let admin = false;  
console.log("Acesso permitido?", logado && admin);
```



OPERADORES



EXERCÍCIO

**CRIE DUAS VARIÁVEIS: TEMINGRESSO E MAIORDEHDADE.
EXIBA SE A PESSOA PODE ENTRAR NO SHOW USANDO O
OPERADOR .**



AGORA VAMOS FALAR DA PARTE DE

CONTROLE DE FLUXO

VAMOS FOCAR EM ESTRUTURAS CONDICIONAIS (IF, ELSE, ELSE IF, SWITCH))



ESTRUTURAS CONDICIONAIS



CONDICIONAIS

- ◆ **ESSA ESTRUTURA PERMITE EXECUTAR BLOCOS DIFERENTES DE CÓDIGO DEPENDENDO DE UMA CONDIÇÃO VERDADEIRA OU FALSA.**

✓ SINTAXE BÁSICA:

```
if (condição) {  
    // executa se condição for verdadeira  
} else if (outraCondição) {  
    // executa se a outra condição for verdadeira  
} else {  
    // executa se nenhuma das condições for verdadeira  
}
```



CONDICIONAIS

EXEMPLO 1: QUESTÃO SIMPLES (VISUAL)

```
let idade = 17;  
if (idade >= 18) {  
    console.log("Maior de idade");  
} else {  
    console.log("Menor de idade");  
}
```



CONDICIONAIS

EXEMPLO 2: USO PRÁTICO

```
let nota = 7;  
  
if (nota >= 7) {  
    console.log("Aprovado");  
}  
else if (nota >= 5) {  
    console.log("Recuperação");  
}  
else {  
    console.log("Reprovado");  
}
```



CONDICIONAIS

EXEMPLO 3: COMBINAÇÃO - EXERCÍCIO

```
let tempo = 60; // em minutos  
  
if (tempo < 30) {  
    console.log("Treino leve");  
}  
else if (tempo <= 60) {  
    console.log("Treino moderado");  
}  
else {  
    console.log("Treino intenso");  
}
```



CONDICIONAIS



EXERCÍCIO

RADAR DO MULTADOR DE RANDAMDAMDAM

- CRIE UMA VARIÁVEL VELOCIDADE.
- SE FOR MENOR QUE 60 → "DENTRO DO LIMITE".
- DE 60 A 80 → "ATENÇÃO" ACIMA.
- DE 80 → "MULTADO".



CONDICIONAIS

- ◆ **FORMA COMPACTA DO IF/ELSE, MUITO USADA EM INTERFACES COM REACT/REACT NATIVE PARA RENDERIZAR CONTEÚDO COM BASE EM CONDIÇÕES.**

- IF TERNÁRIO (OPERADOR CONDICIONAL)**

```
condição ? valor_se_verdadeiro : valor_se_falso
```



CONDICIONAIS

EXEMPLO 1: QUESTÃO SIMPLES (VISUAL)

```
let idade = 20;  
console.log(idade >= 18 ? "Maior de idade" :  
"Menor de idade");
```



CONDICIONAIS

EXEMPLO 2: QUESTÃO PRÁTICA (VISUAL)

```
const usuarioLogado = true;  
const mensagem = usuarioLogado ? "Bem-vindo!" : "Faça login";  
console.log(mensagem);
```



CONDICIONAIS

EXEMPLO 3: USO SIMPLES (VISUAL)

```
let nota = 8;  
let resultado = nota >= 7 ? "Aprovado" : "Reprovado";  
console.log("Resultado:", resultado);
```



CONDICIONAIS



EXERCÍCIO

**CRIE UMA VARIÁVEL PONTUACAO.
SE UM TERNÁRIO PARA IMPRIMIR: "VOCÊ GANHOU UM PRÊMIO!"
SE FOR MAIOR OU IGUAL A 100 "TENTE NOVAMENTE" SE FOR
MENOR.**



AGORA VAMOS VOLTAR A FALAR DA PARTE DE ARAMAZENAMENTO

OBJETOS



OBJETOS

- ◆ UM OBJETO EM JAVASCRIPT É UMA ESTRUTURA QUE ARMAZENA PARES DE CHAVE-VALOR, USADA PARA REPRESENTAR UM ELEMENTO COM PROPRIEDADES. MUITO USADO EM QUALQUER APLICAÇÃO, INCLUSIVE EM REACT/REACT NATIVE, PARA REPRESENTAR USUÁRIOS, PRODUTOS, ESTADOS, CONFIGURAÇÕES, ETC.

✓ SINTAXE BÁSICA:

```
let aluno = {  
    nome: "Ana",  
    idade: 16,  
    aprovado: true  
};  
  
console.log(aluno.nome); // "Ana"
```



AQUI TEMOS UM OBJETO ALUNO COM 3 PROPRIEDADES. PODEMOS ACESSAR OS VALORES USANDO O PONTO (.) .

OBJETOS

✓ EXEMPLO 2: USO PRÁTICO

```
let produto = {  
    nome: "Fone de ouvido",  
    preco: 150,  
    emEstoque: true  
};  
  
console.log("Produto:", produto.nome);  
console.log("Preço: R$", produto.preco);
```

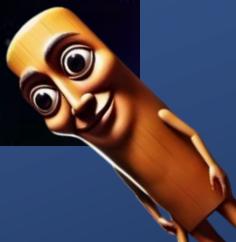
OBJETOS PERMITEM ORGANIZAR DADOS DE FORMA CLARA E AGRUPADA, MUITO COMUM EM APIs, CADASTROS E CONFIGURAÇÕES.

OBJETOS

✓ EXEMPLO 3: COMBINAÇÃO – EXERCÍCIO

```
let livro = {  
    titulo: "JavaScript Essencial",  
    paginas: 240,  
    autor: "João Silva"  
};  
  
console.log(`O ${livro.titulo} tem ${livro.paginas} páginas.`);
```

console.log(`O \${livro.titulo} tem \${livro.paginas} páginas.`);



OBJETOS



✓ EXEMPLO 3: COMBINAÇÃO – EXERCÍCIO

🧠 MODIFICAR OU ADICIONAR PROPRIEDADES:
VOCÊ PODE ALTERAR OU ADICIONAR PROPRIEDADES A UM OBJETO ASSIM:

```
livro.paginas = 300;  
livro.genero = "Programação";  
  
console.log(livro);  
  
console.log(livro)?
```



OBJETOS



EXERCÍCIO

CRIE UM OBJETO CHAMADO CARRO COM AS PROPRIEDADES:

MODELO (STRING)

ANO (NÚMERO)

AUTOMATICO (BOOLEANO)

HMPRIMA UMA FRASE COMO:

"O CARRO MODELO CIVIC DO ANO 2020 É AUTOMÁTICO? TRUE"

OBJETOS



DICA DE USO EM REACT / REACT NATIVE

- 💡 **OBJETOS SÃO FREQUENTEMENTE USADOS PARA REPRESENTAR O ESTADO DE UM COMPONENTE COM USESTATE:**

```
const usuario = {  
  nome: "Carlos",  
  idade: 20  
};  
  
// JSX (React/React Native):  
// <Text>{usuario.nome}</Text>
```



AGORA VAMOS VOLTAR A FALAR DA PARTE DE ARAMAZENAMENTO



ARRAYS

ARRAYS

◆ UM ARRAY É UMA LISTA ORDENADA DE VALORES EM JAVASCRIPT.
CADA VALOR (OU "ITEM") POSSUI UM ÍNDICE NUMÉRICO COMEÇANDO DO 0.
ARRAYS SÃO IDEIAS PARA REPRESENTAR COLEÇÕES: LISTAS DE NOMES, PRODUTOS, NOTAS, TAREFAS, ETC.

✓ SINTAXE BÁSICA:

```
let frutas = ["maçã", "banana", "uva"];
console.log(frutas[0]); // "maçã"
```



AQUI CRIAMOS UM ARRAY COM 3 FRUTAS. ACESSAMOS O PRIMEIRO ITEM USANDO O ÍNDICE 0.

ARRAYS

✓ EXEMPLO 2: USO PRÁTICO

```
let notas = [7.5, 8.0, 9.2];
let media = (notas[0] + notas[1] + notas[2]) / 3;

console.log("Média das notas:", media.toFixed(2));
```



ARRAYS PERMITEM AGRUPAR DADOS SIMILARES (COMO NOTAS) E REALIZAR OPERAÇÕES FACILMENTE.

O MÉTODO .TOFIXED FORMATA UM NÚMERO COM N CASAS DECIMais, RETORNANDO UMA STRING.

ARRAYS

✓ EXEMPLO 3: USO PRÁTICO

```
let tarefas = ["Lavar roupa", "Estudar JS", "Ir ao mercado"];  
  
console.log("Tarefas de hoje:");  
console.log("- " + tarefas[0]);  
console.log("- " + tarefas[1]);  
console.log("- " + tarefas[2]);
```



ARRAYS



EXERCÍCIO

**CRIE UM ARRAY CHAMADO LINGUAGENS COM 3 LINGUAGENS DE PROGRAMAÇÃO QUE VOCÊ CONHECE.
IMPRIMA UMA FRASE COMO: "MINHAS LINGUAGENS FAVORITAS SÃO: JAVASCRIPT, PYTHON, FLUTTER"**



ARRAYS



OPERAÇÕES ÚTEIS COM ARRAYS:

ADICIONAR ITEM AO FINAL:

```
frutas.push("laranja");
```

LER TAMANHO DO ARRAY:

```
console.log(frutas.length); // 4
```

LER TAMANHO DO ARRAY:

```
console.log(frutas[frutas.length - 1]);
```



ARRAYS



OPERAÇÕES ÚTEIS COM ARRAYS:



push()

```
let lista = ["a", "b"];
lista.push("c");
console.log(lista); // ["a", "b", "c"]
```

ADICIONA UM ELEMENTO AO FINAL DO ARRAY.



ARRAYS



OPERAÇÕES ÚTEIS COM ARRAYS:



`pop()`

```
let lista = ["a", "b", "c"];
lista.pop();
console.log(lista); // ["a", "b"]
```

REMOVE O ÚLTIMO ELEMENTO DO ARRAY.

ARRAYS



OPERAÇÕES ÚTEIS COM ARRAYS:

REMOVE O PRIMEIRO ELEMENTO DO ARRAY.

```
let fila = ["primeiro", "segundo", "terceiro"];
fila.shift();
console.log(fila); // ["segundo", "terceiro"]
```

ADICIONA UM OU MAIS ELEMENTOS NO INÍCIO DO ARRAY.

```
let fila = ["b", "c"];
fila.unshift("a");
console.log(fila); // ["a", "b", "c"]
```

ARRAYS



OPERAÇÕES ÚTEIS COM ARRAYS:

RETORNA A QUANTIDADE DE ELEMENTOS NO ARRAY.

```
let numeros = [1, 2, 3];
console.log(numeros.length); // 3
```

**VERIFICA SE UM VALOR ESTÁ PRESENTE NO ARRAY.
RETORNA TRUE OU FALSE.**

```
let frutas = ["maçã", "banana"];
console.log(frutas.includes("banana")); // true
```



ARRAYS



OPERAÇÕES ÚTEIS COM ARRAYS:

RETORNA O ÍNDICE DE UM ITEM. SE NÃO EXISTIR, RETORNA -1

```
let cores = ["azul", "verde", "amarelo"];
console.log(cores.indexOf("verde")); // 1
```

**JUNTA TODOS OS ELEMENTOS DO ARRAY EM UMA STRING,
SEPARADOS POR UM CARACTERE. JAVASCRIPT COPIAR EDITAR**

```
let palavras = ["Eu", "amo", "JS"];
console.log(palavras.join(" ")); // "Eu amo JS"
```



ARRAYS



OPERAÇÕES ÚTEIS COM ARRAYS:

RETORNA UMA PARTE DE UM ARRAY, SEM MODIFICAR O ORIGINAL.

```
let numeros = [1, 2, 3, 4, 5];
console.log(numeros.slice(1, 3)); // [2, 3]
```

**ADICIONA, SUBSTITUI OU REMOVE ELEMENTOS DO ARRAY.
JAVASCRIPT COPIAR EDITAR**

```
let lista = ["a", "b", "d"];
lista.splice(2, 0, "c"); // adiciona "c" na posição 2
console.log(lista); // ["a", "b", "c", "d"]
```



ARRAYS



OPERAÇÕES ÚTEIS COM ARRAYS:

EXECUTA UMA FUNÇÃO PARA CADA ITEM DO ARRAY (NÃO RETORNA NADA).

```
let nomes = ["Ana", "Carlos", "João"];
nomes.forEach(nome => {
  console.log("Olá, " + nome);
});
```

CRIA UM NOVO ARRAY COM BASE NA TRANSFORMAÇÃO DE CADA ITEM.

```
let numeros = [1, 2, 3];
let dobrados = numeros.map(n => n * 2);
console.log(dobrados); // [2, 4, 6]
```



ARRAYS



OPERAÇÕES ÚTEIS COM ARRAYS:

CRIA UM NOVO ARRAY COM ELEMENTOS QUE PASSAM UMA CONDIÇÃO.

```
let notas = [7, 4, 8, 5];
let aprovados = notas.filter(n => n >= 6);
console.log(aprovados); // [7, 8]
```

RETORNA O PRIMEIRO ELEMENTO QUE SATISFAZ A CONDIÇÃO.

```
let usuarios = [
  { nome: "Ana", ativo: false },
  { nome: "Carlos", ativo: true }
];

let ativo = usuarios.find(user => user.ativo);
console.log(ativo.nome); // "Carlos"
```



ARRAYS

OPERAÇÕES ÚTEIS COM ARRAYS:

ORDENA OS ELEMENTOS DO ARRAY EM ORDEM ALFABÉTICA OU NUMÉRICA.

```
let numeros = [10, 2, 5];
numeros.sort((a, b) => a - b); // ordenação crescente
console.log(numeros); // [2, 5, 10]
```



ESTRUTURAS DE REPETIÇÃO EM JAVASCRIPT

Repeticao



ESTRUTURAS DE REPETIÇÃO

- ◆ ESTRUTURAS DE REPETIÇÃO (OU "LAÇOS") SÃO USADAS PARA EXECUTAR UM BLOCO DE CÓDIGO VÁRIAS VEZES DE FORMA AUTOMÁTICA, ATÉ QUE UMA CONDIÇÃO SEJA ATINGIDA.

PODEMOS PRENDER A LEITURA E A EXECUÇÃO DO CÓDIGO EM UM PONTO ESPECÍFICO.

ESTRUTURAS DE REPETIÇÃO

- ◆ WHILE: EXECUTA O BLOCO ENQUANTO A CONDIÇÃO FOR VERDADEIRA.

✓ EXEMPLO 1: VISUAL E SIMPLES

```
let contador = 1;

while (contador <= 3) {
    console.log("Repetição", contador);
    contador++;
}

}
```



 **GLOSSÁRIO:**
i++ é a mesma coisa de i +1

ESTRUTURAS DE REPETIÇÃO

- ◆ WHILE: EXECUTA O BLOCO ENQUANTO A CONDIÇÃO FOR VERDADEIRA.

✓ EXEMPLO 2: COMBINAÇÃO – EXERCÍCIO

```
let tentativas = 0;
let acerto = false;

while (!acerto && tentativas < 3) {
    tentativas++;
    console.log(`Tentativa ${tentativas}`);
    if (tentativas === 2) acerto = true;
}
console.log("Fim do processo");

//console.log(`...fim do bloco...`);
```

ARRAYS



EXERCÍCIO

CRIE UM CÓDIGO QUE EXIBA OS NÚMEROS DE 5 A 1 USANDO WHILE. AO FINAL, MOSTRE "CONTAGEM REGRESSIVA ENCERRADA".



ESTRUTURAS DE REPETIÇÃO

- ◆ **FOR: USADO QUANDO VOCÊ SABE QUANTAS VEZES QUER REPETIR ALGO.**

✓ EXEMPLO 1: VISUAL E SIMPLES

```
for (let i = 1; i <= 3; i++) {  
    console.log("Repetição", i);  
}
```

ESTRUTURAS DE REPETIÇÃO

- ◆ **FOR: USADO QUANDO VOCÊ SABE QUANTAS VEZES QUER REPETIR ALGO.**

✓ EXEMPLO 2: VISUAL E SIMPLES

```
let notas = [7, 8, 6];

for (let i = 0; i < notas.length; i++) {
    console.log("Nota:", notas[i]);
}
```

```
}
```

ESTRUTURAS DE REPETIÇÃO

- ◆ **FOR: USADO QUANDO VOCÊ SABE QUANTAS VEZES QUER REPETIR ALGO.**

✓ EXEMPLO 3: COMBINAÇÃO – EXERCÍCIO

```
for (let i = 10; i >= 1; i--) {  
    console.log("Contagem:", i);  
}  
console.log("Lançar foguete!");
```

console.log("Lançar foguete!");



ARRAYS



EXERCÍCIO

CREIE UM **FOR QUE EXIBA TODOS OS NÚMEROS PARES DE 0 A 10.**



ESTRUTURAS DE REPETIÇÃO

- ◆ **FOR OF: SERVE PARA PERCORRER ARRAYS (OU STRINGS) DE FORMA SIMPLES, SEM PRECISAR USAR ÍNDICES MANUALMENTE.**

✓ EXEMPLO 1: VISUAL E SIMPLES

```
let nomes = ["Ana", "Carlos", "João"];

for (let nome of nomes) {
    console.log("Olá,", nome);
}

}
```

ESTRUTURAS DE REPETIÇÃO

- ◆ **FOR OF: SERVE PARA PERCORRER ARRAYS (OU STRINGS) DE FORMA SIMPLES, SEM PRECISAR USAR ÍNDICES MANUALMENTE.**

✓ EXEMPLO 2: VISUAL E SIMPLES

```
let numeros = [2, 4, 6, 8];

let soma = 0;

for (let numero of numeros) {
    soma += numero;
}

console.log("Soma total:", soma);
```

ESTRUTURAS DE REPETIÇÃO

- ◆ **FOR OF: SERVE PARA PERCORRER ARRAYS (OU STRINGS) DE FORMA SIMPLES, SEM PRECISAR USAR ÍNDICES MANUALMENTE.**

✓ EXEMPLO 3: COMBINAÇÃO – EXERCÍCIO

```
let tarefas = ["Estudar", "Limpar a casa", "Ir ao mercado"];

for (let tarefa of tarefas) {
  console.log("Tarefa pendente:", tarefa);
}
```



ARRAYS



EXERCÍCIO

**CRIE UM ARRAY CURSOS COM 3 CURSOS QUE VOCÊ GOSTA.
USE FOR...OF PARA EXIBIR:
"ESTOU MATRICULADO EM: O NOME DO CURSO"**



ESTRUTURAS DE REPETIÇÃO

- ◆ **FOR IN: SERVE PARA PERCORRER AS PROPRIEDADES (CHAVES) DE UM OBJETO.**

✓ EXEMPLO 1: VISUAL E SIMPLES

```
let aluno = {  
    nome: "Lucas",  
    idade: 17,  
    turma: "3A"  
};  
  
for (let chave in aluno) {  
    console.log(chave); // nome, idade, turma  
}
```

ESTRUTURAS DE REPETIÇÃO

✓ EXEMPLO 2: VISUAL E SIMPLES

```
let carro = {  
    modelo: "Fusca",  
    cor: "azul",  
    ano: 1980  
};  
  
for (let propriedade in carro) {  
    console.log(propriedade + ":", carro[propriedade]);  
}  
}
```

ESTRUTURAS DE REPETIÇÃO

✓ EXEMPLO 3: COMBINAÇÃO – EXERCÍCIO

```
let livro = {  
    titulo: "Aprendendo JS",  
    paginas: 220,  
    autor: "Fernanda"  
};  
  
for (let campo in livro) {  
    console.log(` ${campo.toUpperCase()}: ${livro[campo]}`);  
}  
  
}  
console.log(` ${campo.toUpperCase()}: ${livro[campo]}`);
```



ARRAYS



EXERCÍCIO

**CREIE UM OBJETO CHAMADO FILME COM AS PROPRIEDADES
TITULO, ANO, DIRETOR.**

**USE FOR...IN PARA EXIBIR TODAS AS INFORMAÇÕES DO FILME NO
CONSOLE.**





FUNÇÕES



FUNÇÕES

- ◆ CONCEITO: FUNÇÕES SÃO BLOCOS DE CÓDIGO REUTILIZÁVEIS QUE EXECUTAM UMA TAREFA. ELAS RECEBEM VALORES (PARÂMETROS), EXECUTAM ALGO, E PODEM RETORNAR UM VALOR.

FUNÇÕES

◆ FUNÇÃO COM FUNCTION (FORMA TRADICIONAL)

✓ EXEMPLO 1: VISUAL E SIMPLES

```
function nomeDaFuncao(parametro) {  
    // bloco de código  
    return resultado;  
}
```

FUNÇÕES

◆ FUNÇÃO COM FUNCTION (FORMA TRADICIONAL)

✓ EXEMPLO 1: VISUAL E SIMPLES

```
function digaOlá() {  
    console.log("Olá!");  
}  
  
digaOlá(); // Chamada da função
```

FUNÇÕES

✓ EXEMPLO 2: COMBINAÇÃO – EXERCÍCIO

```
function somar(a, b) {  
    return a + b;  
}  
  
let resultado = somar(5, 3);  
console.log("Resultado:", resultado);
```

console.log("Resultado:", resultado);

FUNÇÕES

✓ EXEMPLO 3: COMBINAÇÃO – EXERCÍCIO

```
function verificarIdade(idade) {  
    if (idade >= 18) {  
        return "Maior de idade";  
    } else {  
        return "Menor de idade";  
    }  
}  
  
console.log(verificarIdade(20));
```

```
console.log(verificarIdade(50));
```



FUNÇÕES



EXERCÍCIO

**CRIE UMA FUNÇÃO CHAMADA CALCULARMÉDIA QUE RECEBA 3 NOTAS E RETORNE A MÉDIA.
USE A FUNÇÃO E EXIBA A FRASE:
"A MÉDIA DO ALUNO É: R"**



FUNÇÕES

◆ FUNÇÕES ANÔNIMAS (ARMAZENADAS EM VARIÁVEIS)

```
const dobro = function(n) {  
    return n * 2;  
}  
  
console.log(dobro(4)); // 8
```

```
console.log(dobro(4)); // 8
```

FUNÇÕES

◆ ARROW FUNCTIONS - SINTAXE MAIS CURTA, MUITO USADA EM REACT/NODE.

✓ EXEMPLO 1: VISUAL E SIMPLES

```
const saudacao = () => {
  console.log("Bem-vindo!");
};
```

```
const quadrado = x => x * x;
console.log(quadrado(5)); // 25
```

FUNÇÕES

◆ FUNÇÃO COM MÚLTIPLOS PARÂMETROS E RETURN

```
const apresentar = (nome, idade) => {  
  return `Olá, meu nome é ${nome} e tenho ${idade} anos.`;  
};
```

```
console.log(apresentar("Ana", 21));
```

```
console.log(apresentar("Ana", 21));
```



FUNÇÕES



EXERCÍCIO

CRIE UMA ARROW FUNCTION DESCONTO QUE RECEBA DOIS VALORES: PREÇO E PERCENTUAL DE DESCONTO. RETORNE O VALOR COM O DESCONTO APLICADO.
EXEMPLO: DESCONTO(100, 20) → DEVE RETORNAR 80





LEITURA E ESCRITA DE ARQUIVOS JSON

JSON

- ◆ **OBJETIVO: LER E MODIFICAR DADOS SALVOS EM UM ARQUIVO .JSON, SIMULANDO UM PEQUENO BANCO DE DADOS LOCAL.**



JSON

- ◆  **1. PREPARAÇÃO DO PROJETO**

```
mkdir meuApp  
cd meuApp  
npm init -y
```

JSON

- ◆ ✓ **CRIE UM ARQUIVO DADOS.JSON COM CONTEÚDO INICIAL:**

```
[  
  { "id": 1, "nome": "Ana", "idade": 17 },  
  { "id": 2, "nome": "Lucas", "idade": 18 }  
]
```



JSON

- ◆ ✓ **CREIE UM ARQUIVO INDEX.JS:**

```
type nul > index.js
```



JSON

- ◆ ✓ **IMPORTAR O MÓDULO FS (FILE SYSTEM)**

```
const fs = require('fs');
```



◆ ✓ LEITURA DO ARQUIVO JSON

```
const dados = fs.readFileSync('dados.json', 'utf-8');
const usuarios = JSON.parse(dados);

console.log("Usuários cadastrados:");
console.log(usuarios);
```

console.log(usuarios);

- ◆ O `readFileSync` lê o conteúdo do arquivo.
- ◆ O `JSON.parse()` transforma o texto em um array de objetos.



◆ ✓ ADICIONAR NOVO USUÁRIO

```
usuarios.push({ id: 3, nome: "Carlos", idade: 19 });

fs.writeFileSync('dados.json', JSON.stringify(usuarios, null, 2));
console.log("Novo usuário adicionado!");
```

com o que? (novo usuário adicionado)?

- ◆ O JSON.stringify() transforma o array em texto.
- ◆ null, 2 deixa o JSON formatado bonitinho.



JSON

- ◆ ✓ SIMULANDO ENTRADA DO USUÁRIO (COM PROMPT)
INSTALE PROMPT-SYNC:

```
npm install prompt-sync
```



JSON

- ◆ ✓ CÓDIGO COM ENTRADA:

```
const prompt = require('prompt-sync')();
const fs = require('fs');

let dados = fs.readFileSync('dados.json', 'utf-8');
let usuarios = JSON.parse(dados);

const nome = prompt("Digite o nome: ");
const idade = parseInt(prompt("Digite a idade: "));

const novoUsuario = {
    id: usuarios.length + 1,
    nome,
    idade
};

usuarios.push(novoUsuario);
fs.writeFileSync('dados.json', JSON.stringify(usuarios, null, 2));

console.log("Usuário cadastrado com sucesso!");
```

JSON



◆ 🔍 OBSERVAÇÕES DIDÁTICAS:

JSONPARSE() → **TRANSFORMA O CONTEÚDO LIDO EM UM OBJETO MANIPULÁVEL**

JSONSTRINGIFY() → **TRANSFORMA O OBJETO DE VOLTA EM TEXTO PARA SALVAR**

FS.READFILESYNC E FS.WRITEFILESYNC → **FAZEM LEITURA E ESCRITA NO DISCO**

PROMPT-SYNC → **PERMITE SIMULAR UM MENU NO TERMINAL**

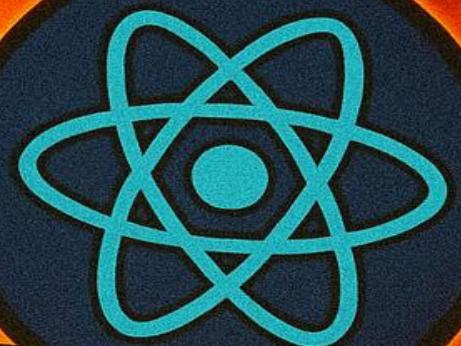


JSON
React
Native

JS

{}

{...}

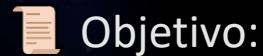




DESAFIO FINAL

🥊 Missão Final: Desafie o Código!

Monte um sistema de cadastro de alunos que nunca desliga!



Objetivo:

Construa um programa com as seguintes opções de navegação:

[1] Cadastrar aluno

- O programa deve solicitar o nome e a idade do aluno
- Os dados devem ser salvos em um JSON

[2] Ver lista

- Mostre todos os alunos cadastrados lendo diretamente do JSON

[3] S Sair

- Finaliza o programa