

Aula: Fundamentos da Web e Introdução ao Front-end com JS

J por José Paulo





Objetivos da Aula

1

Aprender JS

Conhecer a estrutura básica do Javascript.

2

Praticar

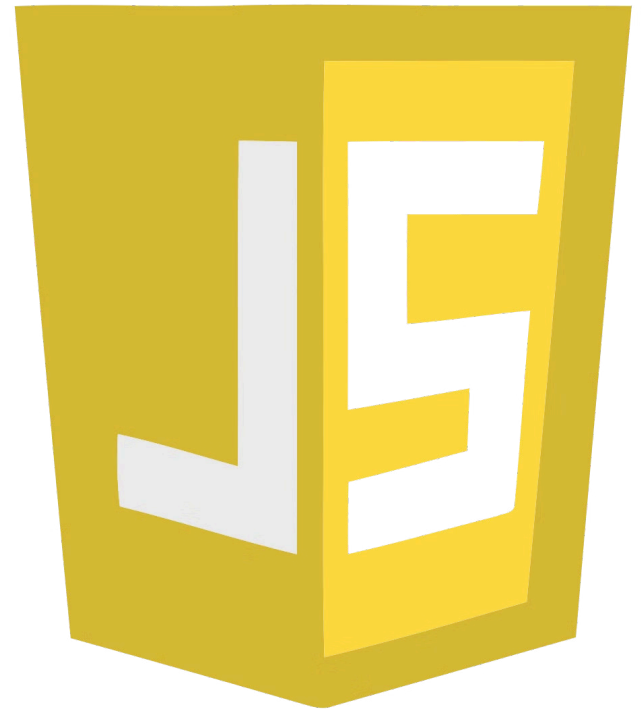
Aplicar JS para reforçar lógica de programação.

O que é JS (Javascript)?

- JavaScript ou JS é uma linguagem de programação interpretada
- Juntamente com HTML e CSS, o JavaScript é uma das três principais tecnologias WEB
- É uma linguagem amplamente usada em navegadores web (client-side)
- É também usada em servidores através de Node.js (exemplo)
- É linguagem multi-paradigma com suporte a estilos de programação orientados a eventos, funcionais e imperativos (orientado a objetos e prototype-based)
- É baseada em ECMAScript *, padronizada pela Ecma international nas especificações ECMA-262[6] e ISO/IEC 16262
- ...e tem **nada a ver** com Java!

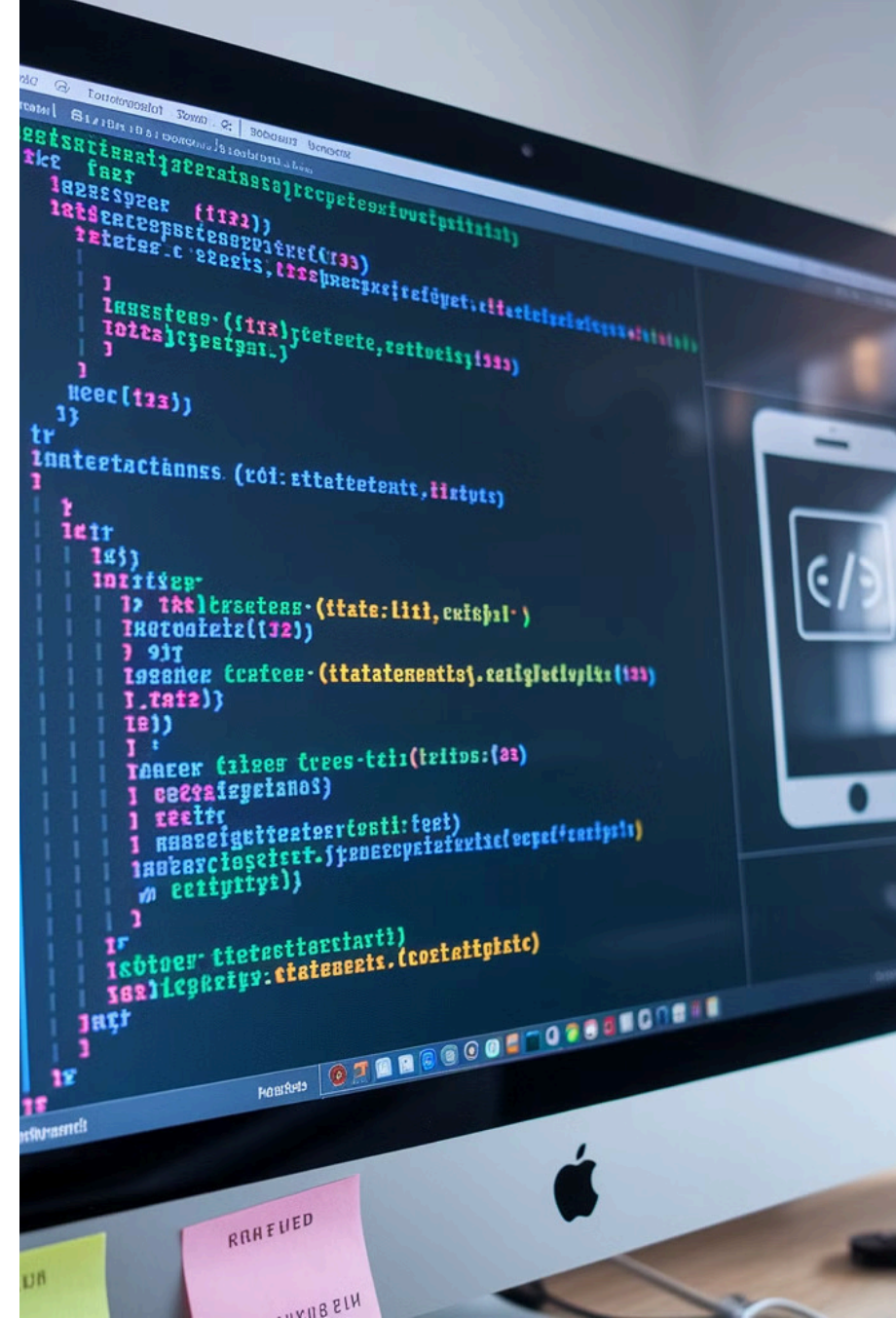
⚠ Site para verificar a ultima versão do [ECMAScript](#)

JavaScript




Mas para que serve o JavaScript no Front end?

- Interação com elementos de uma página HTML ([DOM](#))
- Trabalhar com variáveis, resultados e lógica
- Proporcionar interações ricas ao usuário
- Requisitar dados e informações do servidor sem recarregar a página (AJAX/fetch API)
- Desenvolver aplicativos mobile (PhoneGap, IONIC e React Native)




Como usar JS no documento HTML?

Tag `<script>`

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Script Interno</title>
  <script>
    console.log('Este é um script interno');
  </script>
</head>
  <body>
    <!-- TAGS DO MEU SITE -->
     </body>
</html>
```


Como usar JS no documento HTML?

Script Externo (Arquivo .js Separado)

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Script Interno</title>
  <script src="script.js" defer></script>
</head>
  <body>
    <!-- TAGS DO MEU SITE -->
     </body>
</html>
```

Qual o melhor lugar para colocar a tag `script`

✓ Antes do fechamento do `</body>`

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Script Interno</title>
  </head>
  <body>
    <!-- TAGS DO MEU SITE -->
    <script src="script.js"></script>
     </body>
</html>
```

Qual o melhor lugar para colocar a tag script

✓ No <head> com defer

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Script Interno</title>
    <script src="script.js" defer></script>
  </head>
  <body>
    <!-- TAGS DO MEU SITE -->
  </body>
</html>
```




O defer faz o script rodar somente após o carregamento do HTML, sem bloquear a página.

⚠ Evite usar <script> no <head> sem defer, pois pode bloquear o carregamento da página.

Como declarar uma variável?

JavaScript possui três formas de declarar variáveis: `var`, `let` e `const`. Cada uma tem regras específicas de escopo, reatribuição e hoisting (sempre vamos trabalhar com `let` ou `const`).

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Script Interno</title>
    <script >
      var minhaVariavel = "Olá, Mundo!";
      let minhaVariavel2 = "Olá, Mundo!";
      const minhaVariavel3 = "Olá, Mundo!";

      console.log(minhaVariavel);
      console.log(minhaVariavel2);
      console.log(minhaVariavel3);
    </script>
  </head>
  <body>
    <!-- TAGS DO MEU SITE -->
     </body>
</html>
```

Qual usar?

Recomendações para declaração de variáveis em JavaScript:



Use **const** sempre que possível

A melhor opção para valores que não precisam ser alterados.



Use **let** se precisar reatribuir o valor

Ideal para variáveis que mudarão durante a execução do código.



Evite **var**

Pode causar bugs difíceis de encontrar devido ao seu comportamento de escopo.

⚠ Site para se aprofundar na diferença entre elas: [Here](#)

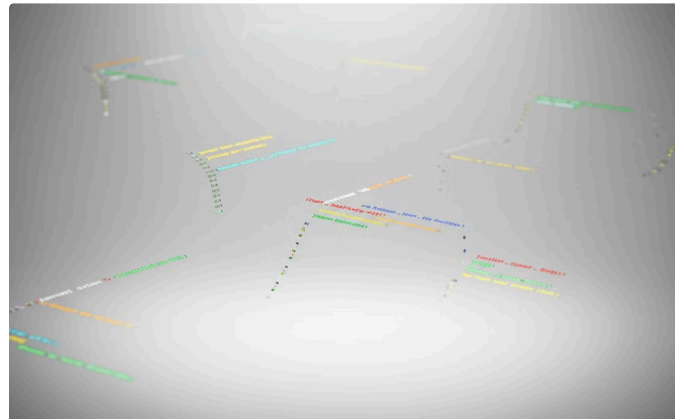
Funções

Funções em JavaScript são blocos de código reutilizáveis que podem ser chamados para executar tarefas específicas.

```
function sayHello (name) {  
  console.log('Hello ' + name);  
}  
  
sayHello('Fonzie');  
  
function sayHello (name) {  
  console.log('Hello ' + name);  
}  
  
sayHello('Fonzie');
```

Estrutura de Funções

Blocos de código delimitados por chaves { } que podem receber parâmetros e retornar valores.



Reutilização de Código

Uma função pode ser chamada múltiplas vezes em diferentes partes do código, evitando repetição.



Organização do Código

Funções ajudam a modularizar o código, tornando-o mais legível e fácil de manter.

Funções

1. Declaração de Função

Você pode declarar uma função usando a palavra-chave `function`, seguida pelo nome da função, parâmetros (opcionais) entre parênteses e um bloco de código entre chaves.

```
function saudacao(nome) {  
    console.log("Olá, " + nome + "!");  
}  
  
saudacao("João"); // Saída: Olá, João!
```

Funções

2. Funções Anônimas

Funções também podem ser atribuídas a variáveis, sem um nome específico. Essas funções são chamadas de funções anônimas.

```
const saudacao = function(nome) {  
    console.log("Olá, " + nome + "!");  
};  
  
saudacao("Maria"); // Saída: Olá, Maria!
```


Funções

3. Funções de Flecha (Arrow Functions)

As funções de flecha oferecem uma sintaxe mais curta e são frequentemente usadas em expressões de funções.

```
const saudacao = (nome) => {  
    console.log("Olá, " + nome + "!");  
};  
  
saudacao("Carlos"); // Saída: Olá, Carlos!
```

⚠ Para quem deseja se aprofundar na diferença entre arrow function e a function tradicional: [Here](#)

Tipos de dados

String (Texto) - Representa uma sequência de caracteres.

✓ Aspas simples ou duplas podem ser usadas.

```
const nome = "JavaScript";  
const mensagem = 'Olá, mundo!';
```

Number (Números) - Pode ser um número inteiro ou decimal (float).

```
const idade = 25;  
const temperatura = 36.5;
```

Tipos de dados

Boolean (Verdadeiro ou Falso) - Usado para valores lógicos.

✓ Útil em estruturas condicionais (if, while, etc.).

```
const isTrue = true;  
const isFalse = false;
```

Undefined (Indefinido) - Significa que uma variável foi declarada, mas **ainda não recebeu um valor**.

```
let valor;  
console.log(valor); // undefined
```

Tipos de dados

Null (Nulo) - Representa a ausência de valor intencional.

✓ Diferente de `undefined`, pois `null` é atribuído manualmente.

```
const valor = null;  
console.log(valor); // null
```

Array (Lista de Valores) - Um `array` armazena vários valores em uma única variável.

✓ Principais Métodos:

- `push("Laranja")` → Adiciona no final
- `pop()` → Remove do final
- `shift()` → Remove do início
- `map()`, `filter()`, `reduce()`, `forEach()` → Manipulação funcional

```
const frutas = ["Maçã", "Banana", "Uva"];  
console.log(frutas[0]); // "Maçã"
```

Tipos de dados

Objeto (Estrutura Chave-Valor) - Um **objeto** armazena dados como pares **chave: valor**.

✓ Objetos podem conter funções (métodos) e serem manipulados dinamicamente.

```
const pessoa = {  
  nome: "Lucas",  
  idade: 30,  
  falar: function () {  
    console.log("Olá, meu nome é " + this.nome);  
  }  
};  
  
console.log(pessoa.nome); // "Lucas"  
pessoa.falar(); // "Olá, meu nome é Lucas"
```

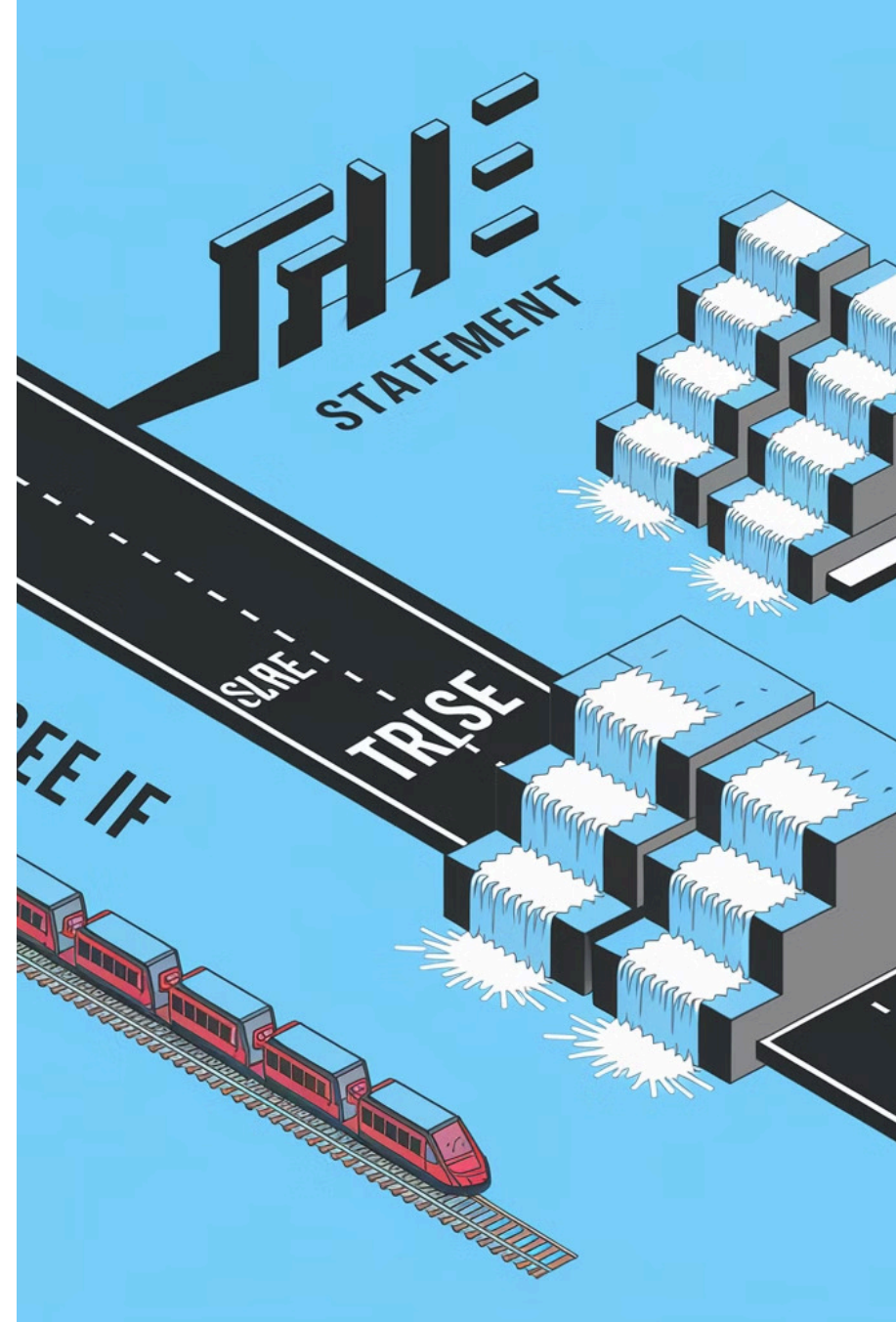
Classe (Molde para Criar Objetos) - É um modelo para criar objetos com as mesmas propriedades e métodos.

✓ Classes são úteis para organização e reuso de código.

```
class Carro {  
  constructor(marca, modelo) {  
    this.marca = marca;  
    this.modelo = modelo;  
  }  
  
  detalhes() {  
    return `Carro: ${this.marca} ${this.modelo}`;  
  }  
}  
  
let meuCarro = new Carro("Toyota", "Corolla");  
console.log(meuCarro.detalhes()); // "Carro: Toyota Corolla"
```


Estruturas condicionais

As estruturas condicionais são usadas para tomar decisões no código, ou seja, executar um bloco de código com base em uma condição. As principais estruturas condicionais em JavaScript são `if`, `else`, `else if` e `switch`.



Estruturas condicionais (if)

A instrução `if` executa um bloco de código se a condição for verdadeira.

```
const idade = 18;  
if (idade ≥ 18) {  
    console.log("Você é maior de idade.");  
}
```

Estruturas condicionais (else)

O `else` é usado para definir um bloco de código alternativo a ser executado quando a condição do `if` não for verdadeira.

```
const idade = 16;  
if (idade ≥ 18) {  
    console.log("Você é maior de idade.");  
} else {  
    console.log("Você é menor de idade.");  
}
```

Estruturas condicionais (else if)

O else if permite testar várias condições, de forma encadeada, quando a condição do if inicial não é verdadeira.

```
const idade = 20;  
if (idade < 18) {  
    console.log("Você é menor de idade.");  
} else if (idade ≥ 18 && idade < 65) {  
    console.log("Você é um adulto.");  
} else {  
    console.log("Você é idoso.");  
}
```

Estruturas condicionais (switch)

O switch é usado quando se tem várias condições possíveis, oferecendo uma sintaxe mais limpa e organizada do que encadear vários if e else if.

```
const cor = "verde";
switch (cor) {
  case "vermelho":
    console.log("A cor é vermelha.");
    break;
  case "azul":
    console.log("A cor é azul.");
    break;
  case "verde":
    console.log("A cor é verde.");
    break;
  default:
    console.log("Cor desconhecida.");
}
```


Estruturas de repetição

As estruturas de repetição são usadas para executar um bloco de código várias vezes, até que uma condição seja satisfeita. Em JavaScript, as principais estruturas de repetição são `for`, `while`, `do...while` e `for...of`, `for...in`.



Estruturas de repetição (for)

O `for` é uma estrutura de repetição muito usada quando se sabe o número de iterações antecipadamente. Ela é composta por três partes: inicialização, condição e incremento.

```
for (let i = 0; i < 5; i++) {  
    console.log(i); // Saída: 0, 1, 2, 3, 4  
}
```

Estruturas de repetição (while)

O `while` executa o bloco de código enquanto a condição for verdadeira. Ele é mais útil quando o número de repetições não é conhecido antecipadamente.

```
let i = 0;
while (i < 5) {
  console.log(i); // Saída: 0, 1, 2, 3, 4
  i++;
}
```

Estruturas de repetição (for...of)

O `for...of` é usado para iterar sobre valores de objetos iteráveis (como arrays, strings, mapas, etc.). Ele facilita a iteração em arrays ou outros objetos iteráveis.

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

for (let fruta of frutas) {
  console.log(fruta); // Saída: maçã, banana, laranja
}
```

Resumo

- **Estruturas Condicionais:** Usadas para tomar decisões no código.
 - if, else, else if: Testa uma condição e executa um bloco de código com base nela.
 - switch: Alternativa para várias condições com valores distintos.
- **Estruturas de Repetição:** Usadas para executar um bloco de código repetidamente.
 - for: Quando se sabe o número de iterações.
 - while: Quando se repete enquanto uma condição for verdadeira.
 - for..of: Itera sobre valores de objetos iteráveis.

Resumo das Comparações

Os **operadores de comparação** em JavaScript são usados para comparar dois valores e retornar um valor booleano (**true** ou **false**), indicando se a comparação é verdadeira ou falsa. Esses operadores são essenciais em estruturas condicionais e de repetição para tomar decisões no código.

| Operador | Descrição | Exemplo | Resultado |
|--------------------|---|------------------------|--------------------|
| <code>==</code> | Igualdade (com conversão de tipo) | <code>"5" == 5</code> | <code>true</code> |
| <code>===</code> | Igualdade estrita (sem conversão de tipo) | <code>"5" === 5</code> | <code>false</code> |
| <code>!=</code> | Diferença (com conversão de tipo) | <code>5 != "5"</code> | <code>false</code> |
| <code>!==</code> | Diferença estrita (sem conversão de tipo) | <code>5 !== "5"</code> | <code>true</code> |
| <code>></code> | Maior que | <code>5 > 3</code> | <code>true</code> |
| <code><</code> | Menor que | <code>3 < 5</code> | <code>true</code> |
| <code>>=</code> | Maior ou igual a | <code>5 >= 5</code> | <code>true</code> |
| <code><=</code> | Menor ou igual a | <code>3 <= 5</code> | <code>true</code> |

 Para quem deseja se aprofundar na diferença entre `===` e `==`: [Here](#)

Desafio Completo de JavaScript (4pts)

Objetivo: Criar uma aplicação simples que utilize variáveis, funções, escopo, desestruturação, operadores de comparação, tipos de dados e manipulação de arrays/objetos. A aplicação deve simular uma "calculadora de tarefas".

1. **Declaração de Variáveis e Tipos:**

- Declare variáveis para armazenar um **nome de usuário** e a **idade**.
- Declare uma variável para armazenar uma **lista de tarefas** (um array de objetos), onde cada tarefa tem:
 - `descricao` (string)
 - `completa` (booleano, `true` ou `false`)
- Declare uma variável `limiteTarefas` que armazena o número máximo de tarefas permitidas.

2. **Funções:**

- Crie uma função chamada `saudarUsuario` que recebe o nome do usuário e a idade e imprime "Olá, [nome], você tem [idade] anos".
- Crie uma função chamada `adicionarTarefa` que adiciona uma nova tarefa à lista de tarefas, verificando se o número de tarefas não excede o `limiteTarefas`.
- Crie uma função chamada `marcarComoCompleta` que marca uma tarefa como completa.
- Crie uma função chamada `listarTarefas` que imprime todas as tarefas na tela (seja completa ou não).
- Crie uma função chamada `removerTarefa` que remove uma tarefa da lista usando a `descricao`.

3. **Operações de Comparação:**

- Verifique se a quantidade de tarefas no array é maior do que o limite (`limiteTarefas`). Se for, mostre uma mensagem dizendo "Você atingiu o limite de tarefas."
- Dentro da função `marcarComoCompleta`, verifique se a tarefa existe antes de tentar marcá-la como completa.

4. **Desestruturação de Objetos:**

- Na função `listarTarefas`, utilize desestruturação para acessar a `descricao` e o `completa` de cada tarefa e imprima uma mensagem formatada, como "Tarefa: [descricao] | Status: [completa]".

5. **Desestruturação de Arrays:**

- Dentro da função `adicionarTarefa`, se a lista de tarefas for menor que o limite, extraia os primeiros dois elementos da lista de tarefas usando desestruturação e mostre-os na tela.

Requisitos

1. A estrutura de dados para armazenar as tarefas deve ser um **array de objetos**.
2. A aplicação deve imprimir informações detalhadas no console, incluindo mensagens de erro quando necessário (por exemplo, ao tentar adicionar mais tarefas do que o permitido).
3. Use **desestruturação** para simplificar o código em funções como `listarTarefas`.
4. Comente cada linha de código especificando o que esta acontecendo.

Exemplo da saída esperada:

```
Olá, João, você tem 30 anos!
Tarefa: Estudar | Status: Incompleta
Tarefa: Comprar leite | Status: Completa
Tarefa: Ir à academia | Status: Incompleta
Tarefa: Estudar | Status: Completa
Você atingiu o limite de tarefas.
```