

Introdução ao Javascript

A única linguagem da Web.

- ☐ Introdução
- ☐ Literais, identificadores e variáveis
- ☐ Comentários
- ☐ `let`, `const` e `var`
- ☐ Tipos de dados
- ☐ Operadores e expressões
- ☐ Operadores aritméticos
- ☐ Operador de atribuição
- ☐ Precedência de operadores
- ☐ Strings
- ☐ Números
- ☐ Formatação geral

Palavra-chave	Pode mudar o valor?	Escopo	Uso recomendado
<code>let</code>	✅ Sim	Bloco (<code>{ }</code>)	✅ Mais comum
<code>const</code>	❌ Não	Bloco (<code>{ }</code>)	✅ Quando não muda
<code>var</code>	✅ Sim	Função/global	⚠️ Evitar (legado)

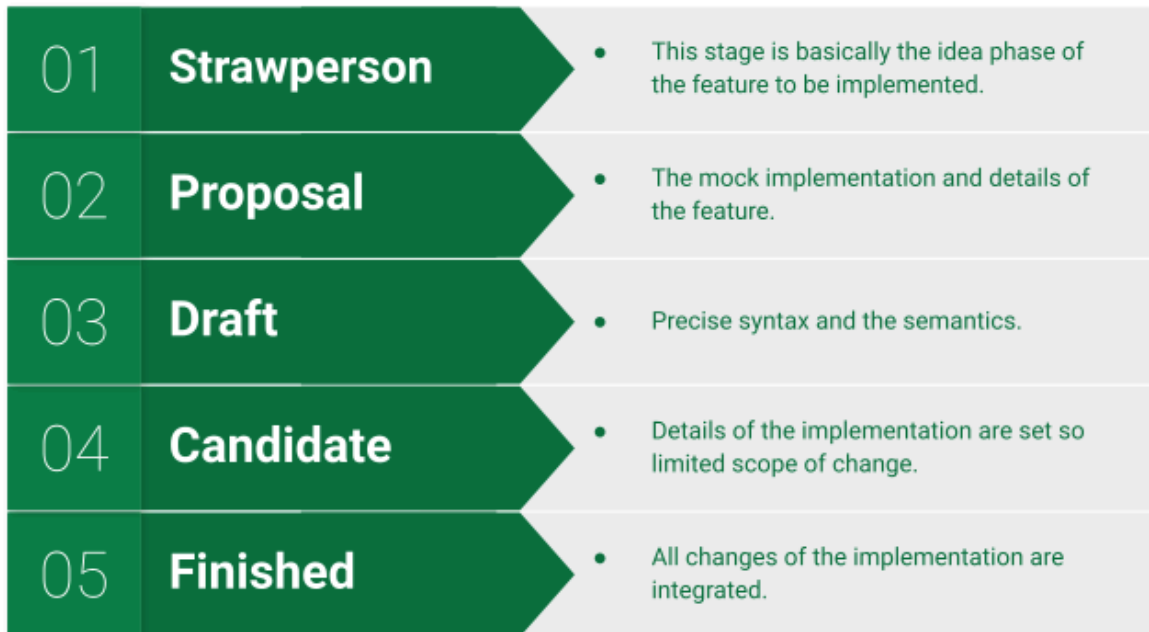
Operador	Ação	Exemplo	Resultado
<code>+</code>	Soma	<code>2 + 3</code>	<code>5</code>
<code>-</code>	Subtração	<code>10 - 4</code>	<code>6</code>
<code>*</code>	Multiplicação	<code>3 * 2</code>	<code>6</code>
<code>/</code>	Divisão	<code>8 / 2</code>	<code>4</code>
<code>%</code>	Resto da divisão	<code>10 % 3</code>	<code>1</code>

Operador	Exemplo	Equivalente a
<code>=</code>	<code>x = 10</code>	Atribuição
<code>+=</code>	<code>x += 5</code>	<code>x = x + 5</code>
<code>-=</code>	<code>x -= 2</code>	<code>x = x - 2</code>
<code>*=</code>	<code>x *= 3</code>	<code>x = x * 3</code>

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

Prioridade (do maior pro menor)	Operadores
1	<code>*</code> <code>/</code> <code>%</code>
2	<code>+</code> <code>-</code>
3	<code>=</code>

The TC39 Process



#1 Variáveis

1. Crie uma variável `cidade` com o valor `"São Paulo"`. Imprima no console.
2. Crie uma constante `anoAtual` com o valor `2025`. Imprima no console.
3. Tente reatribuir o valor de `anoAtual` para `2026` e observe o erro (comente a linha para rodar o código).
4. Crie uma variável com nome inválido (exemplo: `1nome`) e veja o erro (comente a linha para rodar o código).
5. Declare as variáveis primitivas: `nome = "Lucas"`, `idade = 30`, `temConta = true`. Imprima todas.
6. Crie um objeto `usuario` com as propriedades `nome`, `idade` e `temConta` usando os valores das variáveis acima. Imprima o objeto.
7. Copie o objeto `usuario` para uma variável chamada `copiaUsuario`.
8. Altere a propriedade `nome` de `copiaUsuario` para `"Ana"`.
9. Imprima o valor da propriedade `nome` tanto de `usuario` quanto de `copiaUsuario` e explique o resultado.

10. Crie um array com 3 nomes e utilize o método `.push()` para adicionar mais um nome. Imprima o array.

#2 Operadores e Expressões

1. Crie uma variável `num = 8` e imprima seu valor.
2. Crie duas variáveis `a = 15` e `b = 4`. Imprima a soma, subtração, multiplicação, divisão e o resto da divisão entre `a` e `b`.
3. Calcule e imprima o valor da expressão `3 + 5 * 2`. Depois, altere para `(3 + 5) * 2` e imprima o resultado.
4. Crie uma variável `saldo = 200`. Aumente o saldo em 100 usando o operador `+=` e imprima o valor.
5. Diminua o saldo em 50 usando o operador `=` e imprima o valor.
6. Use `++saldo` para incrementar o saldo antes de imprimir.
7. Use `saldo++` para imprimir o saldo e depois incrementá-lo. Imprima o saldo novamente para ver a mudança.
8. Crie duas variáveis de string: `nome = "João"` e `saudacao = "Olá, "`. Imprima a concatenação das duas usando `+`.
9. Declare `let i = 10`. Imprima o valor de `++i`, depois `i++`, e finalmente o valor atual de `i`.
10. Verifique se o saldo é maior que 200 e imprima o resultado (true ou false).

#3 Strings e números

1. Crie uma variável chamada `nome` com o valor `"Maria"`. Imprima no console a frase:
"Olá, Maria!" usando template literals.
2. Crie uma string com seu nome usando aspas simples. Imprima a quantidade de caracteres dessa string.
3. Crie uma string `"JavaScript"` e imprima ela toda em maiúsculas.
4. Verifique se a string `"programação"` contém a substring `"gram"` e imprima o resultado (`true` ou `false`).

5. Crie uma variável `mensagem` com a seguinte string (use template literals para quebra de linha):

```
Olá!  
Seja bem-vindo(a) ao JavaScript.
```

Imprima essa mensagem.

6. Crie uma variável `numero` com o valor `9.87654321`. Imprima o número com apenas 3 casas decimais.
7. Converta o número `123` para uma string e imprima o tipo da variável resultante.
8. Crie duas variáveis: `produto` com o valor `"Teclado"` e `preco` com o valor `159.99`.

Imprima a frase:

```
O produto "Teclado" custa R$ 159,99.
```

Use template literals e o método `toFixed()` para formatar o preço.

9. Concatene as strings `"Bom"` e `"dia"` usando o operador `+` e imprima o resultado.
10. Crie uma string com o valor `" espaço "` (com espaços no início e fim). Use o método `.trim()` para remover os espaços e imprima o resultado.



DOM e APIs globais

Como o navegador enxerga a página web.

- ☐ Introdução
- ☐ DOM - Document Object Model
 - ☐ Tipos de nós
 - ☐ Atravessando o DOM
 - ☐ Editando o DOM
 - ☐ Eventos
- ☐ `window`
- ☐ `localStorage` e `sessionStorage`

```
<html>
  <body>
    <h1>Olá</h1>
    <p>Bem-vindo</p>
```

```
</body>
</html>
```

```
document
├── html
│   └── body
│       ├── h1 ("Olá")
│       └── p ("Bem-vindo")
```

localStorage	sessionStorage
Persiste após fechar o navegador	Limpa ao fechar a aba/janela
Compartilhado entre abas do mesmo domínio	Isolado por aba

Exercícios

Seleção e Estrutura do DOM

1. Identificar nós:

Crie uma página com `<h1>`, `<p>` e um comentário.

No console, use `document.childNodes` e `nodeType` para listar **todos os tipos de nós** existentes na raiz.

2. Primeiro elemento

Exiba no console o **primeiro elemento filho** de `<body>` usando `firstElementChild` e compare com `firstChild`.

3. Coleções vivas vs. estáticas

- Selecione todos os `` com `getElementsByName` e `querySelectorAll`.
- Adicione novos `` via JS e observe qual coleção se atualiza automaticamente.

Navegação e Modificação

4. Próximo e anterior

Crie 3 parágrafos. Use `nextElementSibling` e `previousElementSibling` para percorrer a lista e alterar a cor de cada um em sequência.

5. Alterando conteúdo

Selecione um `<p>` e:

- Troque o texto com `textContent`.
- Adicione uma tag `` no meio usando `innerHTML`.

6. Criando elementos

Ao clicar em um botão, crie dinamicamente um `` com texto digitado em um `<input>` e insira na `` existente.

7. Removendo elementos

Crie um botão "Remover último item" que apaga o último `` da lista usando `remove()` ou `removeChild()`.

Eventos

8. Clique e estilo

Crie um botão que, ao ser clicado, adicione ou remova uma classe `.ativo` em um `<div>` usando `classList.toggle()`.

9. Evento de teclado

Em um `<input>`, exiba em tempo real no console cada tecla pressionada (`keydown`) e mostre também `event.key`.

10. Formulário e `preventDefault`

Monte um formulário com `input` e `submit`.

Ao enviar, use `event.preventDefault()` para impedir o recarregamento da página e exibir os dados em um `<p>`.

API `window` e Timers

11. Viewport

Mostre em um `<p>` a largura e altura da janela (`innerWidth`, `innerHeight`) sempre que ela for redimensionada (`resize`).

12. Alert/Confirm/Prompt

Ao clicar em um botão:

- Use `confirm` para perguntar se o usuário quer continuar.
- Se sim, peça o nome com `prompt` e exiba uma mensagem de boas-vindas.

13. **Contador com** `setInterval`

Crie um contador regressivo de 10 a 0 que atualiza a cada segundo.

Quando chegar a 0, use `clearInterval` para parar e mostre "Tempo esgotado!".

Armazenamento

14. **localStorage**

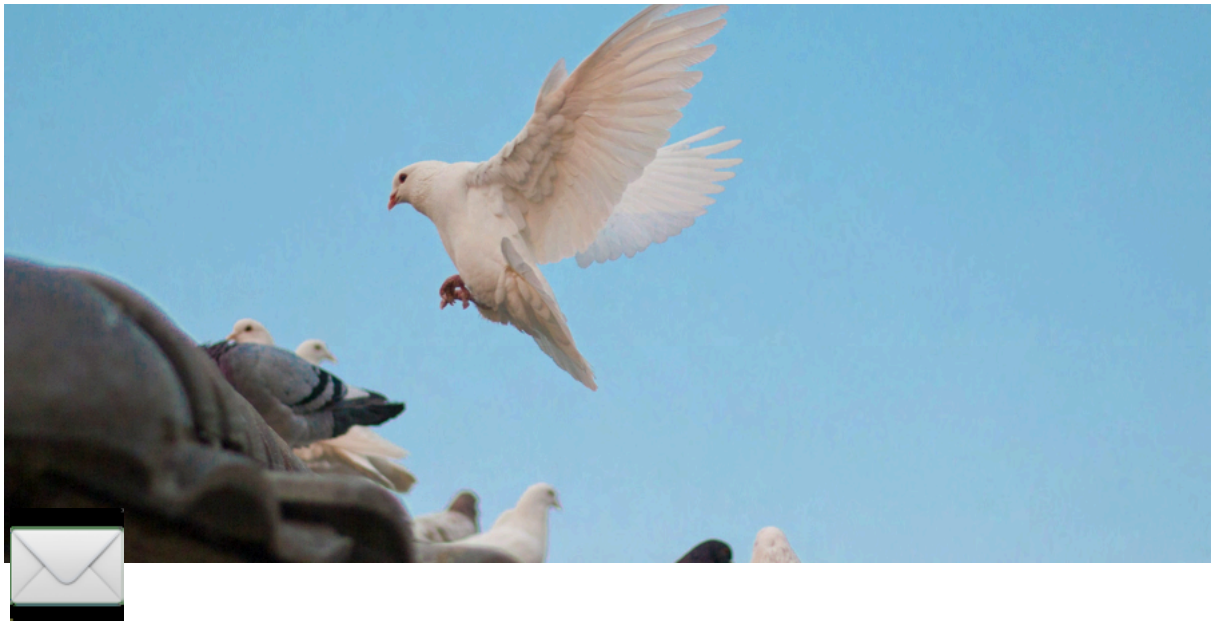
Crie um formulário para salvar o nome do usuário no `localStorage`.

Ao recarregar a página, exiba automaticamente "Olá, [nome]" se já houver valor salvo.

15. **sessionStorage**

Crie um contador de visitas **na aba atual**.

A cada reload, incremente o valor armazenado em `sessionStorage` e exiba "Você recarregou esta aba X vezes".



Funções


Mostrando como fazer e quando fazer

- ☐ Parâmetros
- ☐ Retornando valores
- ☐ Arrow functions

Closures - JavaScript | MDN

Uma closure é a combinação de uma função com as referências ao estado que a circunda (o ambiente léxico). Em outras palavras, uma closure lhe dá acesso ao escopo


 <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Closures>

 mdn web docs

IIFE - Glossário do MDN Web Docs | MDN

IIFE (Immediately Invoked Function Expression) é uma função em JavaScript que é executada assim que definida.

 <https://developer.mozilla.org/pt-BR/docs/Glossary/IIFE>

 mdn web docs

Exercícios

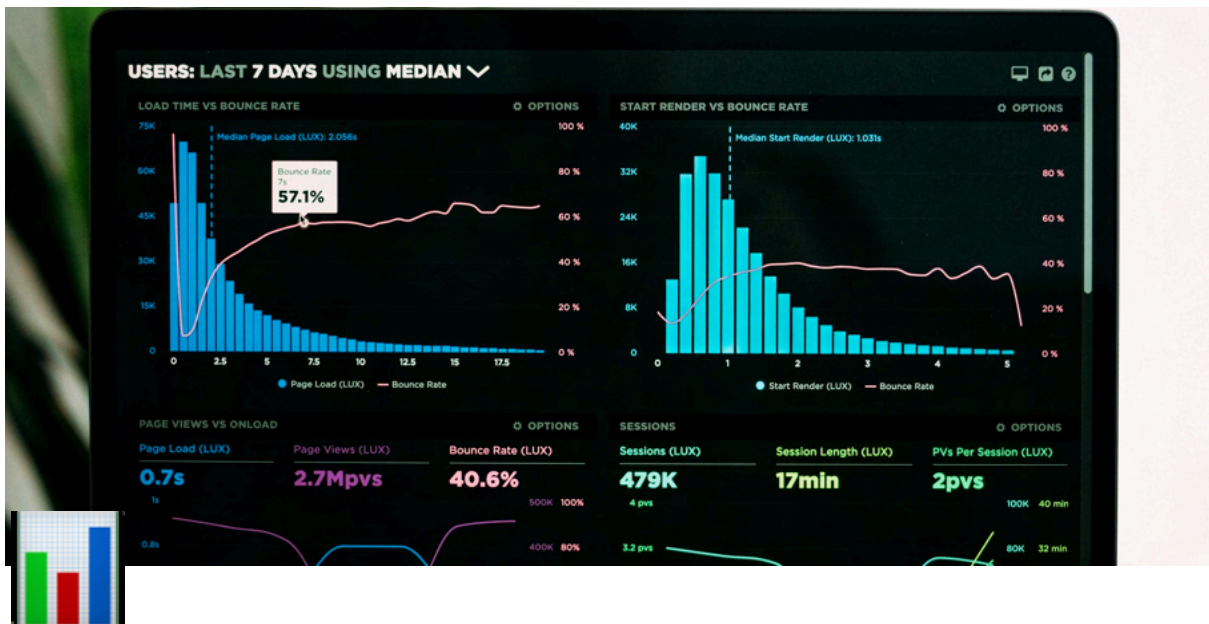
1. Crie uma função chamada `mostrarMensagem` que imprime `"Olá, mundo!"`. Depois, chame essa função.
2. Crie uma função `cumprimentar(nome)` que recebe um nome e imprime `"Olá, <nome>!"`. Teste com seu nome.
3. Crie uma função `cumprimentarOpcional(nome)` onde `nome` tem valor padrão `"Visitante"`. Imprima a saudação e teste chamando a função com e sem argumento.
4. Crie a função `informarPessoa(nome, idade)` que imprime nome e idade. Teste passando argumentos na ordem correta e invertida para observar o resultado.
5. Use a função abaixo para imprimir `"Oi, Lucas!"` passando apenas o segundo argumento.

```
function mostrarMensagem(texto = "Oi", usuario = "Anonimo") {  
  console.log(`${texto}, ${usuario}!`)  
}
```

6. Implemente `criarProduto({ nome, preco, disponivel })` que imprime os dados do produto. Teste omitindo alguns campos.
7. Crie uma função `multiplicar(...numeros)` que retorna o produto de todos os números passados.
8. Crie uma função `infoPessoa(nome, idade, ativo)` que retorna um objeto com essas propriedades e mais uma `status` que indica `"Ativo"` ou `"Inativo"` conforme o booleano.
9. Faça `classificarIdade(idade)` que retorna `"Menor"` se idade < 18, ou `"Adulto"` caso contrário. Coloque um `console.log` após o `return` para ver se executa.
10. Implemente `minMax(array)` que retorna um array com o mínimo e máximo valor do array recebido. Use desestruturação para obter esses valores.
11. Crie `dadosPessoa()` que retorna um objeto com `nome`, `idade` e `ativo`. Use desestruturação para imprimir cada propriedade.
12. Reescreva a função abaixo usando arrow function com sintaxe reduzida:

```
function triplo(n) {  
  return n * 3  
}
```

13. Reescreva a função `triplo(n)` como arrow function, que retorna o triplo de `n`. Se `n` não for número, retorne 0.
14. Crie uma arrow function que recebe dois números e retorna sua soma.
15. Crie uma arrow function `criarCarro(marca, ano)` que retorna um objeto com as propriedades `marca` e `ano`. Desafio: não utilizar `return`.



Arrays I

Manipulando listas de dados.

- ☐ Introdução
- ☐ Distribuir itens com spread
- ☐ Modificando array
- ☐ Matrizes
- ☐ Desestruturando arrays
- ☐ Limites dos arrays

Exercícios

```
const frutas = ["maçã", "banana", "laranja", "uva", "manga", "abacaxi", "pêssago", "kiwi", "melancia", "coco"];
```

1. Copiar o array usando o operador spread
Crie uma cópia do array `frutas` usando o operador spread.

2. Combinar dois arrays usando o operador spread

Combine o array `frutas` com outro array `["morango", "amora", "framboesa"]` usando o operador spread.

3. Desestruturar as três primeiras frutas do array

Use a desestruturação para extrair as três primeiras frutas do array `frutas`.

4. Desestruturar a primeira fruta e colocar o resto em outro array

Extraia a primeira fruta do array `fruta` e coloque o restante das frutas em um novo array.

5. Combinar e desestruturar dois arrays de frutas

Combine `frutas` com outro array `["morango", "amora", "framboesa"]` e depois desestrua para pegar a primeira fruta e o resto das frutas.

6. Trocar frutas usando desestruturação

Troque os valores de duas variáveis extraídas do array `frutas` usando a desestruturação.

7. Passar elementos de um array como argumentos para uma função

Use o operador spread para passar os elementos do array `frutas` como argumentos para uma função que concatena três frutas.

```
function concatenaFrutas(p1, p2, p3) {  
  return `${p1}, ${p2}, ${p3}`;  
}
```

8. Adicionar frutas no início do array

Adicione as palavras `"morango"`, `"amora"` e `"framboesa"` no início do array `frutas` usando o operador spread.

9. Dividir o array em duas partes

Use a desestruturação para separar o array `frutas` em duas partes: as primeiras cinco frutas e o restante.

10. Remover uma fruta do meio do array

Use o operador spread para criar um novo array que exclui a palavra `"manga"` do array `frutas`.



Arrays II

Manipulando dados com métodos nativos.

☐ Funções anônimas em arrays

☐ `map`

☐ `filter`

☐ `sort`

☐ `find` e `findIndex`

☐ `forEach`

☐ `reduce`

☐ Outros métodos

Exercícios

```
const users = [  
  {  
    id: 1,  
    name: "John Doe",
```



```
    email: "john.doe@example.com",
    age: 28,
    isActive: true
  },
  {
    id: 2,
    name: "Jane Smith",
    email: "jane.smith@example.com",
    age: 34,
    isActive: false
  },
  {
    id: 3,
    name: "Michael Johnson",
    email: "michael.johnson@example.com",
    age: 22,
    isActive: true
  },
  {
    id: 4,
    name: "Emily Davis",
    email: "emily.davis@example.com",
    age: 40,
    isActive: false
  },
  {
    id: 5,
    name: "William Brown",
    email: "william.brown@example.com",
    age: 31,
    isActive: true
  }
];
```

map

1. Retornar os nomes dos usuários

Crie um novo array contendo apenas os nomes dos usuários.

2. Retornar os e-mails dos usuários em caixa alta

Crie um novo array com os e-mails dos usuários, mas todos em letras maiúsculas.

3. Aumentar a idade de todos os usuários em 1 ano

Crie um novo array onde a idade de cada usuário é aumentada em 1 ano.

4. Transformar o status ativo em "sim" ou "não"

Crie um novo array que transforma o valor booleano de `isActive` em uma string `"sim"` se for `true` e `"não"` se for `false`.

5. Retornar um array de ids e nomes

Crie um array de objetos contendo apenas os campos `id` e `name` de cada usuário.

filter

1. Filtrar usuários ativos

Crie um novo array contendo apenas os usuários que têm o campo `isActive` como `true`.

2. Filtrar usuários com mais de 30 anos

Crie um novo array com os usuários cuja idade (`age`) é maior que 30.

3. Filtrar usuários com e-mails que contêm "example"

Crie um array contendo apenas os usuários cujo e-mail contém a string `"example"`.

4. Filtrar usuários com nome que começa com "j"

Crie um array contendo apenas os usuários cujos nomes começam com a letra `"j"`.

5. Filtrar usuários com id par

Crie um array com os usuários que possuem um id que é um número par.

sort

1. Ordenar por idade em ordem crescente

Ordenar o array de usuários pela idade em ordem crescente (do mais jovem ao mais velho).

2. Ordenar por idade em ordem decrescente

Ordenar o array de usuários pela idade em ordem decrescente (do mais velho ao mais jovem).

3. Ordenar por nome em ordem alfabética

Ordenar o array de usuários pelos nomes em ordem alfabética.

4. Ordenar por nome em ordem alfabética reversa

Ordenar o array de usuários pelos nomes em ordem alfabética reversa.

5. Ordenar por status ativo, colocando usuários ativos primeiro

Ordenar o array de usuários de forma que os usuários com `isActive` igual a `true` apareçam primeiro.

`find` e `findIndex`

1. Encontrar o primeiro usuário ativo

Use `find()` para localizar o primeiro usuário cujo campo `isActive` é `true`.

2. Encontrar o índice do primeiro usuário ativo

Use `findIndex()` para encontrar o índice do primeiro usuário cujo campo `isActive` é `true`.

3. Encontrar o usuário com um determinado e-mail

Use `find()` para encontrar o usuário que possui um e-mail específico, por exemplo, `"michael.johnson@example.com"`.

4. Encontrar o índice de um usuário com um determinado e-mail

Use `findIndex()` para localizar o índice de um usuário que possui o e-mail `"michael.johnson@example.com"`.

5. Encontrar o primeiro usuário com idade superior a 30 anos e seu índice

Use `find()` para localizar o primeiro usuário com mais de 30 anos.

Use `findIndex()` para obter o índice desse usuário.

`forEach`

1. Exibir os nomes de todos os usuários

Percorrer o array e exibir no console o nome de cada usuário.

2. Exibir o status de atividade de cada usuário

Exibir no console se cada usuário está ativo ou inativo.

3. Somar todas as idades (sem usar `reduce()`)

Calcular a soma das idades de todos os usuários.

4. Adicionar uma propriedade "ano de nascimento" a cada usuário

Adicionar uma nova propriedade `anoDeNascimento` a cada usuário com base em sua idade e o ano atual.

5. Exibir os e-mails de usuários ativos

Exibir no console o e-mail apenas dos usuários que estão ativos (`isActive: true`).

`reduce`

1. Somar as idades dos usuários

Calcular a soma das idades de todos os usuários.

2. Contar o número de usuários ativos

Contar quantos usuários têm o campo `isActive` como `true` .

3. Agrupar usuários por status de atividade

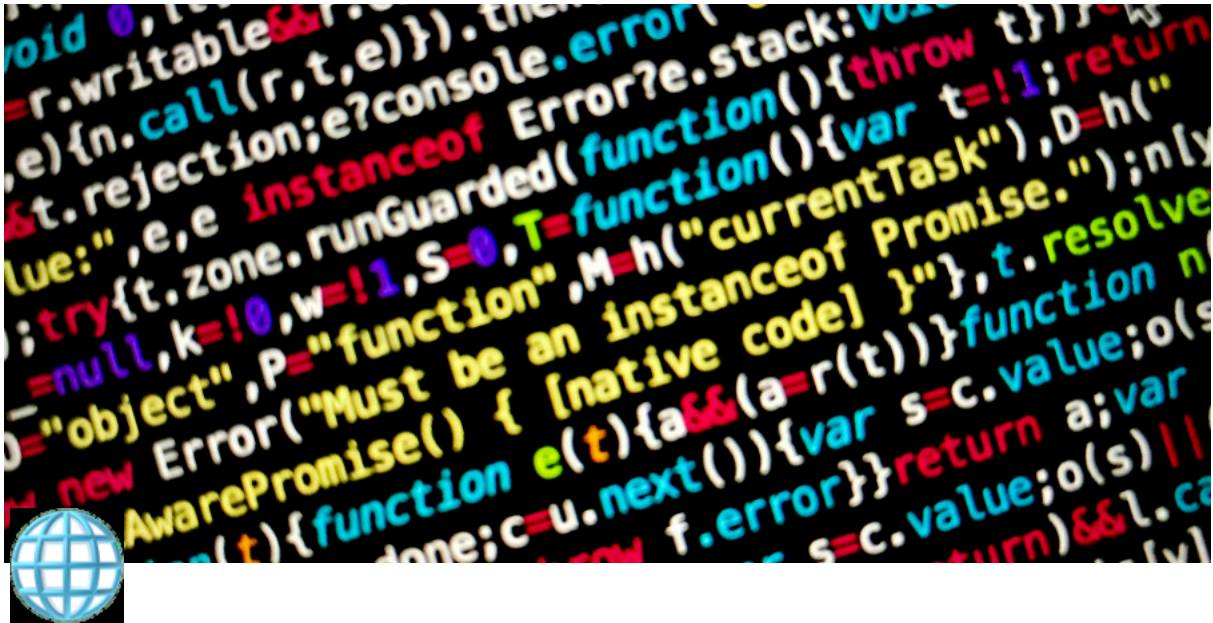
Agrupar os usuários em dois arrays: um para os usuários ativos e outro para os inativos.

4. Encontrar o usuário mais velho

Use `reduce()` para encontrar o usuário com a maior idade no array.

5. Concatenar os nomes dos usuários em uma única string

Criar uma string que contenha todos os nomes dos usuários separados por vírgulas.



Objetos

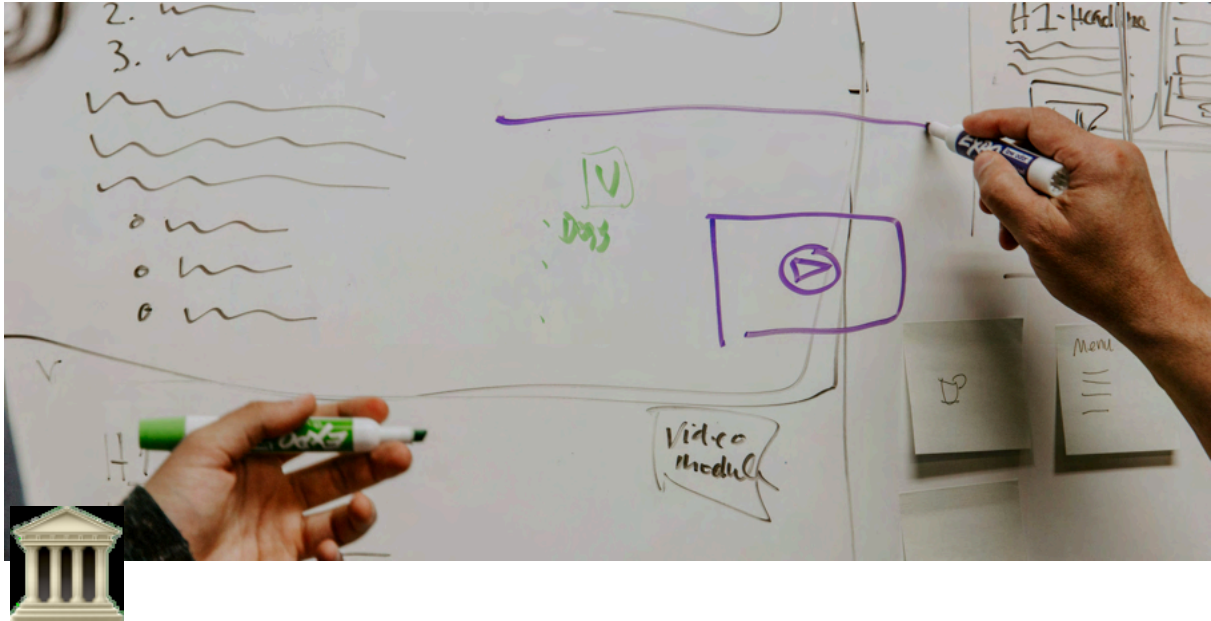
Como os dados em si são formados

- ☐ Introdução
- ☐ Criando objetos
- ☐ Propriedades
- ☐ Métodos
- ☐ Referência vs valor
- ☐ `this`
- ☐ Desestruturação

Exercícios

1. Crie um objeto `book` usando **notação literal** com as propriedades: `title`, `author` e `pages`.
2. Crie um objeto vazio usando `new Object()` e adicione dinamicamente as propriedades `name` e `age`.

3. Crie um objeto `animal` com a propriedade `species`. Depois, crie um novo objeto `dog` com `animal` como protótipo.
4. Crie uma variável `x` com valor 10 e outra `y` que recebe `x`. Altere `y` e verifique o valor de `x`.
5. Crie um objeto `user` com a propriedade `loggedIn: false`. Copie o objeto para `admin` e mude `admin.loggedIn` para `true`. O que acontece com `user.loggedIn`?
6. Crie um array `a` com três números. Copie o array para `b` e altere um elemento de `b`. Verifique se `a` também foi alterado.
7. Crie um objeto `lamp` com uma função chamada `turnOn` que imprime `"Lâmpada ligada"` no console. Execute o método.
8. Crie um objeto `player` com propriedades `name` e `score`, e um método `showScore` que imprime uma mensagem usando `this.name` e `this.score`.
9. Crie um objeto `robot` com uma propriedade `model` e um método `identify` usando **arrow function**. Veja o valor de `this.model`.
10. Crie um objeto `movie` com as propriedades `title`, `year` e `genre`. Use desestruturação para extrair `title` e `year`.
11. Use o mesmo objeto `movie` e extraia `genre` para uma variável chamada `type`.
12. Crie uma função que receba um objeto com propriedades `width` e `height`, e retorne a área. Use desestruturação no parâmetro.
13. Crie um objeto `counter` com a propriedade `value` e um método `increment()` que aumenta o valor.
14. Crie dois objetos `cat` e `dog` com propriedade `sound`, e um método `makeSound` em `cat` que imprime `this.sound`. Copie esse método para `dog` e execute.
15. Crie um objeto `original`, copie para `clone`, depois modifique o `clone`. Use `console.log` para mostrar que ambos foram afetados.



ES Modules

- ☐ Introdução
- ☐ `import` e `export`
- ☐ arquivos `.mjs`
- ☐ Default exports
- ☐ Múltiplos exports
- ☐ Renomeando exports
- ☐ Usando a tag `script`



Promises

A espera do que não vemos

☐ Você já foi a um concerto?

Básico

```
let promise = new Promise(function (resolve, reject) {  
  // executor (o código que produz, "cantor")  
});
```

☐ A função "executora"

☐ `resolve` e `reject`

☐ Estados: `pending`, `fulfilled` e `rejected`

☐ Consumindo Promises

☐ `then`, `catch` e `finally`

Encadeamento de *Promises*

- ☐ *Handlers* das Promises
- ☐ O que é e não é considerado encadeamento
- ☐ Retornando Promise a partir do `then`
- ☐ Encadeando requests

Lidando com erros

- ☐ `catch` como fallback
- ☐ O `try..catch` implícito
- ☐ Criando novos erros e encadeando o `catch`

API das *Promises*

- ☐ `all()`
- ☐ `allSettled()`
- ☐ `race()`
- ☐ `any()`

Async/Await

- ☐ Uma alternativa para lidar com Promises.
- ☐ Erros no `await`

The Modern JavaScript Tutorial

Modern JavaScript Tutorial: simple, but detailed explanations with examples and tasks, including: closures, document and events, object oriented programming and

 <https://javascript.info/>



JAVASCRIPT.INFO
The Modern JavaScript Tutorial

Exercícios

Promises Básicas

1. Crie uma Promise que resolva com o valor `"Olá mundo"` após 2 segundos.
2. Crie uma Promise que rejeite com o erro `"Algo deu errado"` após 1 segundo.
3. Utilize `.then()` e `.catch()` para consumir uma Promise que resolve com o número `10`. Multiplique o valor por 5.
4. Crie uma função que retorna uma Promise que resolve com um número aleatório entre 1 e 100 após 1 segundo.
5. Encadeie 3 `.then()` para transformar o número: dobre, adicione 10, divida por 2.
6. Crie uma função que recebe um nome e retorna uma Promise que resolve com `"Olá, {nome}"` após 1 segundo.
7. Simule um erro proposital na segunda etapa de um `.then()`. Use `catch()` para capturá-lo.
8. Use `finally()` para exibir `"Operação concluída"` independente do sucesso ou erro de uma Promise.
9. Use uma função que retorna uma Promise com base em um `setTimeout()` e simule um "carregando..." antes de resolver.
10. Crie uma Promise que verifica se um número é par. Resolva se for par, rejeite se for ímpar.

Fetch com `then` e `catch`

11. Use `fetch()` para buscar `https://dummyjson.com/products/1`. Exiba o nome do produto no console.
12. Crie uma função `getProduct(id)` que retorna uma Promise com os dados de um produto específico da API.
13. Faça 3 requisições em sequência (produto 1, 2 e 3), usando encadeamento de `.then()`.
14. Tente buscar um produto que não existe (ex: `id = 9999`) e trate o erro com `.catch()`.
15. Use `fetch()` para buscar `https://dummyjson.com/users/1` e exibir nome e e-mail.
16. Crie uma função que recebe um `userId` e retorna o nome completo do usuário (`firstName lastName`).

17. Use o método `.text()` da resposta do fetch e mostre o conteúdo cru da resposta.
 18. Tente usar `.json()` duas vezes na mesma resposta. O que acontece? Teste e explique no código.
 19. Encadeie dois fetchs: um para `products/1` e o segundo para `users/1`, mostrando o nome do produto e o usuário.
 20. Simule uma falha no `fetch` alterando a URL para `https://dummyjson.com/erro`. Trate com `catch()`.
-

Async/Await

21. Refaça o exercício 11 usando `async/await`.
22. Crie uma função `getUserNome(id)` que usa `await fetch` para retornar o nome do usuário.
23. Use `try/catch` com `await fetch` para lidar com um erro de requisição (ex: URL incorreta).
24. Faça uma função `getProductPrice(id)` que retorna apenas o preço de um produto.
25. Crie uma função que retorna um array de títulos de 3 produtos diferentes, usando `await` em cada chamada.
26. Use `await` dentro de um loop `for` para buscar os produtos de id 1 a 3, um por vez.
27. Crie uma função `loadProduct(id)` que mostra "Carregando..." no console, depois o produto.
28. Crie uma função `getUserAvatar(id)` que busca a foto de um usuário (`image`) e a exibe no DOM (HTML).
29. Crie uma função que faça uma requisição com `await fetch`, mas jogue um erro proposital com `throw new Error`.
30. Use `await fetch` para obter um usuário. Se a idade for menor que 30, jogue erro. Trate com `try/catch`.

```
// PROMISES BÁSICAS (1-10)
```

```
// 1.
```

```
new Promise(resolve => setTimeout(() => resolve("Olá mundo"), 2000));
```

```
// 2.
```

```
new Promise((_, reject) => setTimeout(() => reject("Algo deu errado"), 1000));
```

```
// 3.
```

```
Promise.resolve(10)  
  .then(num => num * 5)  
  .then(console.log)  
  .catch(console.error);
```

```
// 4.
```

```
function randomNumber() {  
  return new Promise(resolve => setTimeout(() => resolve(Math.floor(Math.random() * 100) + 1), 1000));  
}
```

```
// 5.
```

```
Promise.resolve(5)  
  .then(n => n * 2)  
  .then(n => n + 10)  
  .then(n => n / 2)  
  .then(console.log);
```

```
// 6.
```

```
function greet(name) {  
  return new Promise(resolve => setTimeout(() => resolve(`Olá, ${name}`), 1000));  
}
```

```
// 7.
```

```
Promise.resolve(10)  
  .then(n => n * 2)  
  .then(() => { throw new Error("Erro proposital"); })
```

```

    .then(console.log)
    .catch(console.error);

// 8.
Promise.resolve("OK")
    .then(console.log)
    .catch(console.error)
    .finally(() => console.log("Operação concluída"));

// 9.
function delayedMessage() {
    console.log("Carregando...");
    return new Promise(resolve => setTimeout(() => resolve("Pronto!"), 1000));
}

// 10.
function isEven(num) {
    return new Promise((resolve, reject) => {
        num % 2 === 0 ? resolve("Par") : reject("Ímpar");
    });
}

// FETCH COM THEN/CATCH (11-20)

// 11.
fetch("https://dummyjson.com/products/1")
    .then(res => res.json())
    .then(data => console.log(data.title));

// 12.
function getProduct(id) {
    return fetch(`https://dummyjson.com/products/${id}`).then(res => res.json());
}

// 13.
fetch("https://dummyjson.com/products/1")
    .then(res => res.json())

```

```

.then(data => {
  console.log(data.title);
  return fetch("https://dummyjson.com/products/2");
})
.then(res => res.json())
.then(data => {
  console.log(data.title);
  return fetch("https://dummyjson.com/products/3");
})
.then(res => res.json())
.then(data => console.log(data.title));

// 14.
fetch("https://dummyjson.com/products/9999")
.then(res => {
  if (!res.ok) throw new Error("Produto não encontrado");
  return res.json();
})
.catch(console.error);

// 15.
fetch("https://dummyjson.com/users/1")
.then(res => res.json())
.then(data => console.log(data.firstName, data.email));

// 16.
function getFullName(userId) {
  return fetch(`https://dummyjson.com/users/${userId}`)
    .then(res => res.json())
    .then(data => `${data.firstName} ${data.lastName}`);
}

// 17.
fetch("https://dummyjson.com/users/1")
.then(res => res.text())
.then(console.log);

// 18.

```

```
fetch("https://dummyjson.com/users/1")
  .then(res => {
    res.json().then(console.log);
    res.json().then(console.log); // Erro: body já foi usado
  });
```

// 19.

```
fetch("https://dummyjson.com/products/1")
  .then(res => res.json())
  .then(prod => {
    console.log("Produto:", prod.title);
    return fetch("https://dummyjson.com/users/1");
  })
  .then(res => res.json())
  .then(user => console.log("Usuário:", user.firstName));
```

// 20.

```
fetch("https://dummyjson.com/erro")
  .then(res => res.json())
  .catch(console.error);
```

// ASYNC/AWAIT (21–30)

// 21.

```
async function fetchProduct() {
  const res = await fetch("https://dummyjson.com/products/1");
  const data = await res.json();
  console.log(data.title);
}
```

// 22.

```
async function getUsername(id) {
  const res = await fetch(`https://dummyjson.com/users/${id}`);
  const user = await res.json();
  return user.firstName;
}
```

// 23.

```

async function fetchWithError() {
  try {
    const res = await fetch("https://dummyjson.com/erro");
    const data = await res.json();
    console.log(data);
  } catch (error) {
    console.error("Erro:", error);
  }
}

// 24.
async function getProductPrice(id) {
  const res = await fetch(`https://dummyjson.com/products/${id}`);
  const data = await res.json();
  return data.price;
}

// 25.
async function getTitles() {
  const titles = [];
  for (let i = 1; i <= 3; i++) {
    const res = await fetch(`https://dummyjson.com/products/${i}`);
    const data = await res.json();
    titles.push(data.title);
  }
  return titles;
}

// 26.
async function fetchSequential() {
  for (let i = 1; i <= 3; i++) {
    const res = await fetch(`https://dummyjson.com/products/${i}`);
    const data = await res.json();
    console.log(data.title);
  }
}

// 27.

```



```
async function loadProduct(id) {  
  console.log("Carregando...");  
  const res = await fetch(`https://dummyjson.com/products/${id}`);  
  const data = await res.json();  
  console.log(data.title);  
}
```

// 28.

```
async function getUserAvatar(id) {  
  const res = await fetch(`https://dummyjson.com/users/${id}`);  
  const user = await res.json();  
  const img = document.createElement("img");  
  img.src = user.image;  
  document.body.appendChild(img);  
}
```

// 29.

```
async function fetchWithCustomError() {  
  const res = await fetch("https://dummyjson.com/products/1");  
  throw new Error("Erro proposital!");  
}
```

// 30.

```
async function getUserWithAgeCheck() {  
  try {  
    const res = await fetch("https://dummyjson.com/users/1");  
    const user = await res.json();  
    if (user.age < 30) throw new Error("Usuário muito jovem");  
    console.log(user.firstName);  
  } catch (err) {  
    console.error(err);  
  }  
}
```



Requisições em JS

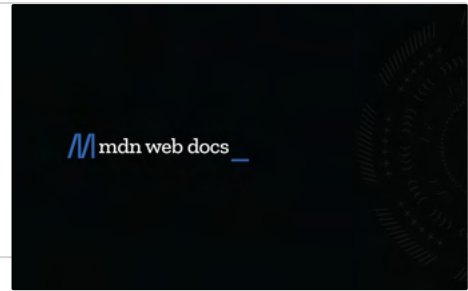
Obtendo conteúdo externo

- ☐ Conceitos de roteamento na Web
 - ☐ Protocolos
 - ☐ IP
 - ☐ URLs
- ☐ Fetch
- ☐ Axios
 - ☐ API
 - ☐ Usando instâncias
 - ☐ Interceptors

Uma visão geral do HTTP - HTTP | MDN

HTTP é um protocolo que permite a obtenção de recursos, como documentos HTML. É a base de qualquer troca de dados na Web e um protocolo cliente-servidor, o que

 <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Guides/Overview>



HyperText Transfer Protocol (HTTP)


Learn about the basic infrastructure of the internet, the HyperText Transport Protocol. The foundation for accessing websites and beyond. With free HTTP testing tools.

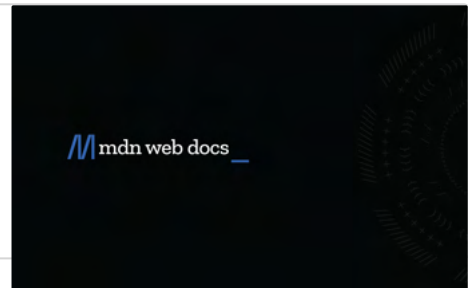
 <https://http.dev/>



What is a URL? - Learn web development | MDN

This article discusses Uniform Resource Locators (URLs), explaining what they are and how they're structured.

 https://developer.mozilla.org/en-US/docs/Learn_web_development/Howto/Web_mechanics/What_is_a_URL#summary



Uniform Resource Identifier (URI)

What is 'Uniform Resource Identifier (URI)'? Discover how to master Uniform Resource Identifier (URI), with free examples and code snippets.

 <https://http.dev/uri>



Introdução | Axios Docs

Axios é um cliente HTTP baseado-em-promessas para o node.js e para o navegador. É isomórfico (= pode rodar no navegador e no node.js com a mesma base de código). No lado do servidor usa o código nativo do node.js - o modulo http, enquanto no lado do cliente (navegador) usa

 <https://axios-http.com/ptbr/docs/intro>



Funções

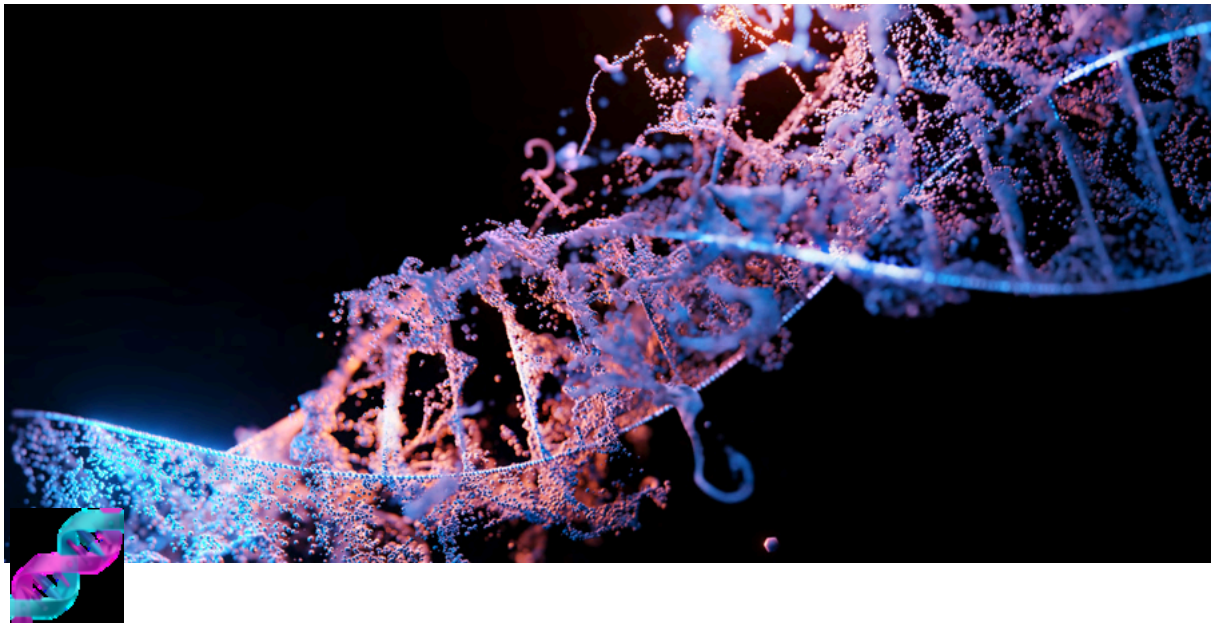
Definindo o que entra e o que sai sistematicamente

- ☐ Os problemas
- ☐ Tipos de parâmetros
- ☐ Parâmetros opcionais
- ☐ Parâmetros padrão
- ☐ Tipo de retorno implícito
- ☐ Tipo de retorno explícito
- ☐ Retorno do tipo `void`
- ☐ Documentação

Exercícios

1. Crie uma função `square` que recebe um número e retorna o quadrado dele. Tipar corretamente o parâmetro e o retorno.

2. Crie uma função `greetUser` que recebe `name` (string opcional) e retorna uma saudação. Se `name` não for passado, usar `"Anonymous"`.
3. Crie uma função `multiply` que recebe dois números e retorna a multiplicação. O segundo parâmetro deve ter valor padrão `1`.
4. Crie uma função `formatValue` que recebe um `value` que pode ser `number` ou `string`. Se for `number`, retorna no formato `"$1,234"`. Se for `string`, retorna em maiúsculas.
5. Crie uma função `logMessage` que recebe uma mensagem (`string`) e apenas imprime no console. Tipo de retorno: `void`.
6. Crie uma função `processInput` que recebe `input: string | number`. Retorna o dobro se for `number`. Retorna a versão em maiúsculas se for `string`.
7. Crie um tipo literal `Direction = 'up' | 'down' | 'left' | 'right'`. Crie uma função `move` que aceita um parâmetro `dir: Direction` e retorna `"Moving ${dir}"`.
8. Crie uma função `introduce` que recebe `name` (obrigatório), `age` (opcional) e `city` (opcional). Retorna uma string do tipo `"Name: John, Age: 30, City: NY"` omitindo valores não fornecidos.
9. Crie uma arrow function `add` que recebe dois números e retorna a soma.
10. Crie uma função `calculateArea` que recebe `width` e `height` e retorna a área de um retângulo. Adicione JSDoc para documentar parâmetros e retorno.



Tipos complexos

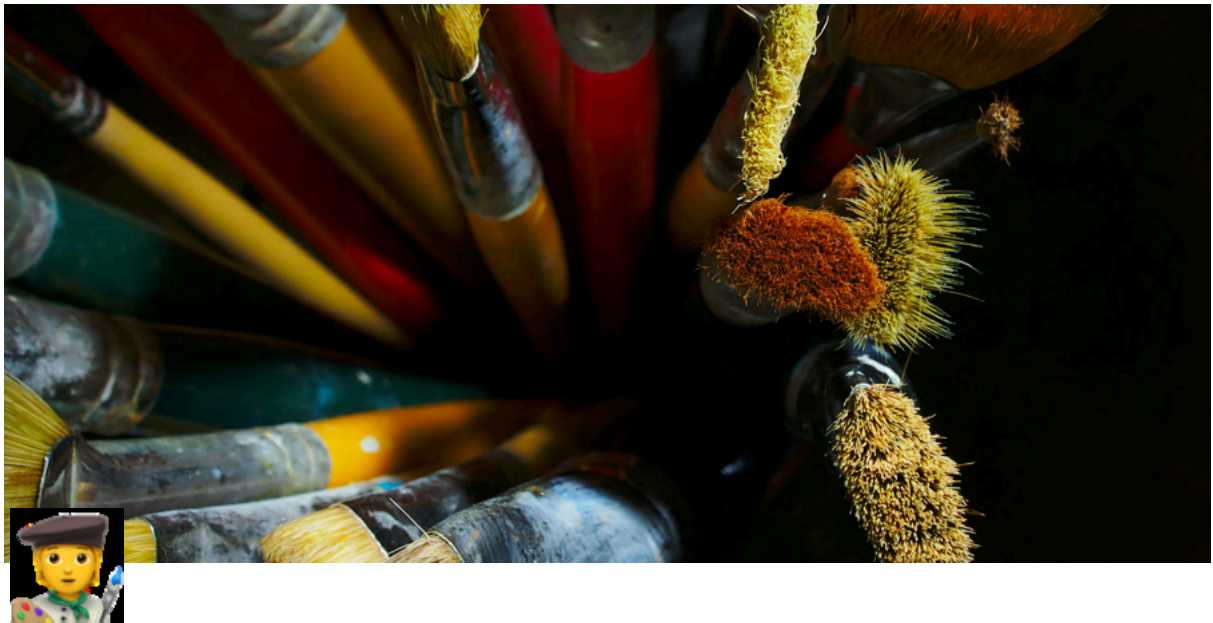
Quando os primitivos não são suficientes

- ☐ Tipos complexos
- ☐ Anotando tipos em um array
- ☐ Arrays multi-dimensionais
- ☐ Tuples
- ☐ Inferência de tipos
- ☐ Parâmetros rest

Exercícios

1. Criar uma função `calculateArea` que receba apenas `'circle'`, `'square'` ou `'rectangle'` e retorne uma mensagem dizendo qual forma foi escolhida.
2. Declarar um tipo `ID` que pode ser `string` ou `number` e criar uma função `printID` que mostre no console o ID recebido.
3. Criar dois tipos `Car` e `Motorcycle`, ambos com `make` e `model`. Escrever uma função que receba um `Car | Motorcycle` e mostre apenas as propriedades comuns.

4. Criar uma função `formatPrice` que aceite números ou textos e retorne o preço formatado com "R\$" na frente.
5. Criar um tipo literal `Direction` com os valores `'up'`, `'down'`, `'left'` e `'right'` e escrever uma função `movePlayer` que mostre a direção escolhida.
6. Criar três tipos `Admin`, `Member` e `Guest`, todos com `username`. Escrever uma função que receba um desses tipos e mostre uma mensagem diferente para cada um.
7. Declarar um array `mixedValues` que contenha números, textos e valores booleanos. Criar uma função que percorra o array e conte quantos elementos de cada tipo existem.
8. Criar uma tupla `loginInfo` com `[username, password, rememberMe]` e escrever uma função que mostre cada valor no console.
9. Criar uma função `combineValues` que receba qualquer quantidade de números ou textos e retorne todos juntos em uma única string.
10. Criar uma função `toggleFeature` que só aceite `'on'`, `'off'` ou `'standby'` e imprima o estado atual da função.



Tipos customizados

Aproveitando a flexibilidade do Typescript

- ☐ Introdução
- ☐ Enums
- ☐ Enums de strings vs Enums numéricos
- ☐ Object Types
- ☐ Type Aliases
- ☐ Function Types
- ☐ Tipos genéricos
- ☐ Funções genéricas

Tipos customizados

1. Crie um enum chamado `Month` representando os 12 meses do ano, começando do número 1 para Janeiro. Declare uma variável `currentMonth` do tipo `Month` e atribua o mês de Maio.

2. Transforme o enum `Month` anterior em um enum de strings, usando letras maiúsculas. Atribua o valor de Dezembro à variável `holidayMonth`.
3. Crie um tipo de objeto `Book` com as propriedades `title` (string), `author` (string) e `pages` (number). Declare uma variável `myBook` usando esse tipo e atribua valores válidos.
4. Crie uma função `printBookInfo` que receba um objeto do tipo `Book` e exiba uma frase descrevendo o livro.
5. Crie um alias `Employee` com as propriedades `name` (string) e `salary` (number). Em seguida, crie um alias `Department` que tenha `manager` do tipo `Employee` e `employees` como array de `Employee`. Declare uma variável `salesDepartment` com dados fictícios.
6. Crie um tipo `StringConcatenator` para funções que recebem dois parâmetros string e retornam uma string. Em seguida, declare uma variável `concatStrings` e atribua uma função compatível que concatene as duas strings com um espaço entre elas.
7. Crie uma função `stringTutor` que receba um callback do tipo `StringConcatenator` e teste a função com duas strings quaisquer, exibindo o resultado no console.
8. Usando o tipo genérico `Family<T>`, declare uma variável `numberFamily` para armazenar números, com dois pais, um cônjuge e três filhos.
9. Utilize a função genérica `getFilledArray<T>` para criar:
 - Um array de 5 strings `'Hello'` chamado `greetingsArray`.
 - Um array de 4 objetos `{ name: string; age: number }` chamado `peopleArray`.
10. Crie um tipo genérico `Box<T>` com a propriedade `content` do tipo `T`. Em seguida, crie uma variável `bookBox` do tipo `Box<Book>` e atribua um objeto contendo um livro.



Tipos de objetos

Tipos para objetos mais complexos

- ☐ Tipos vs interfaces
- ☐ Interfaces e classes
- ☐ Tipos aninhados
- ☐ Tipos compostos
- ☐ `extends`
- ☐ Assinaturas de índices
- ☐ Membros opcionais

Exercícios

1. Criar uma interface `Car` com propriedades `make` (string), `model` (string) e `year` (number). Declarar uma variável `myCar` usando essa interface.
2. Criar uma interface `ElectricCar` que estenda `Car` e adicione a propriedade `batteryCapacity` (number). Declarar uma variável `tesla` usando essa interface.

3. Criar uma interface `Person` com propriedades `name` (string) e `age` (number).
Criar uma interface `Employee` que estenda `Person` e adicione `role` (string).
Declarar um objeto `employee1`.
4. Criar uma interface `Rectangle` com propriedades `width` e `height` (number) e um método `area()` que retorna um número. Implementar um objeto `myRectangle`.
5. Criar um tipo `Point` como objeto com `x` e `y` (números). Declarar uma função `distance` que recebe dois `Point` e retorna a distância entre eles.
6. Criar um tipo `ApiResponse<T>` genérico com `data` (T) e `status` (number).
Declarar uma variável `response` que contenha um array de strings como `data`.
7. Criar uma interface `Product` com `id` (number), `name` (string) e `price` (number).
Criar uma função `discount` que recebe um `Product` e retorna o produto com 10% de desconto.
8. Criar uma interface `Book` com `title` (string), `author` (string) e `pages` (number).
Declarar uma função `printBookInfo` que recebe `Book` e imprime `"Title - Author"`.
9. Criar um tipo `Callback<T>` para funções que recebem um argumento do tipo `T` e retornam void. Declarar uma função `logMessage` que usa `Callback<string>`.
10. Criar uma interface `Team` com `name` (string) e `members` (array de strings).
Declarar uma função `addMember` que recebe `Team` e um `string` e adiciona ao array de membros.



Unões

Combinando o poder de vários tipos

- ☐ Introdução
- ☐ Definindo uniões
- ☐ Estreitando tipos
 - ☐ *Type guards*
 - ☐ Funções
 - ☐ `else`
 - ☐ Estreitando após um *type guard*
- ☐ Inferindo uniões no retorno
- ☐ Uniões e Arrays
- ☐ Valores comuns aos membros
- ☐ Uniões com literais

Exercícios

1. Crie uma variável `userID` que aceite tanto `string` quanto `number`. Atribua primeiro um número e depois uma string.
2. Escreva uma função `logValue` que receba um parâmetro que pode ser `string` ou `number` e apenas faça `console.log` do valor.
3. Crie uma função `formatInput` que receba `string | number` e:
 - se for string, retorne em letras minúsculas (`toLowerCase()`);
 - se for number, retorne com duas casas decimais (`toFixed(2)`).
4. Crie dois tipos `Dog` e `Bird` com métodos específicos: `bark()` e `fly()`.
Escreva uma função `action` que receba `Dog | Bird` e chame o método correto usando `in` como type guard.
5. Crie tipos `Tea` e `Coffee` com métodos `steep()` e `pourOver()`.
Escreva uma função `brew` que use type guard sem `else` para chamar o método correto.
6. Crie uma função `combine` que receba um array de `(string | number)[]` e retorne:
 - strings em maiúsculas;
 - números formatados como moeda (`$`).
7. Crie tipos `Admin` e `Guest` com propriedade `username` e:
 - `Admin` tem `accessLevel`;
 - `Guest` tem `expiryDate`.Escreva uma função `showInfo` que receba `Admin | Guest` e retorne `username`.
8. Crie um tipo literal `Direction` que pode ser `'up' | 'down' | 'left' | 'right'`.
Escreva uma função `movePlayer` que receba `Direction` e faça `console.log` de cada direção.
9. Crie tipos `Circle` e `Rectangle` com método `area()`.
Escreva uma função `calculateArea` que receba `Circle | Rectangle` e retorne o valor da área usando type guard.
10. Crie uma função `handleResponse` que pode receber `string | { status: number; data: string }`.
 - Se for string, apenas faça `console.log`.
 - Se for objeto, faça `console.log` do `status` e do `data`.

▼ Respostas

```
// 1. Variável userID
let userID: string | number;
userID = 123;
userID = "abc123";

// 2. Função logValue
function logValue(value: string | number): void {
  console.log(value);
}
logValue(42);
logValue("Hello");

// 3. Função formatInput
function formatInput(input: string | number): string {
  if (typeof input === "string") {
    return input.toLowerCase();
  } else {
    return input.toFixed(2);
  }
}
console.log(formatInput("HELLO")); // "hello"
console.log(formatInput(3.14159)); // "3.14"

// 4. Função action com Dog | Bird
type Dog = { bark: () => void };
type Bird = { fly: () => void };

function action(animal: Dog | Bird) {
  if ("bark" in animal) {
    animal.bark();
  } else {
    animal.fly();
  }
}

const dog: Dog = { bark: () => console.log("Woof!") };
const bird: Bird = { fly: () => console.log("Flap!") };
```

```

action(dog);
action(bird);

// 5. Função brew com Tea | Coffee
type Tea = { steep: () => void };
type Coffee = { pourOver: () => void };

function brew(drink: Tea | Coffee) {
  if ("steep" in drink) drink.steep();
  if ("pourOver" in drink) drink.pourOver();
}
const tea: Tea = { steep: () => console.log("Steeping tea") };
const coffee: Coffee = { pourOver: () => console.log("Pouring coffee") };
brew(tea);
brew(coffee);

// 6. Função combine
function combine(arr: (string | number)[]): (string | string)[] {
  return arr.map(item =>
    typeof item === "string" ? item.toUpperCase() : `$$${item.toFixed(2)}$`
  );
}
console.log(combine(["apple", 5, "banana", 10])); // ["APPLE", "$5.00", "BANANA", "$10.00"]

// 7. Função showInfo com Admin | Guest
type Admin = { username: string; accessLevel: number };
type Guest = { username: string; expiryDate: Date };

function showInfo(user: Admin | Guest): string {
  return user.username;
}
const admin: Admin = { username: "Alice", accessLevel: 10 };
const guest: Guest = { username: "Bob", expiryDate: new Date() };
console.log(showInfo(admin)); // "Alice"
console.log(showInfo(guest)); // "Bob"

```

```

// 8. Função movePlayer com Direction
type Direction = "up" | "down" | "left" | "right";
function movePlayer(direction: Direction) {
  console.log(`Moving ${direction}`);
}
movePlayer("up");
movePlayer("left");

// 9. Função calculateArea com Circle | Rectangle
type Circle = { radius: number; area: () => number };
type Rectangle = { width: number; height: number; area: () => number };

function calculateArea(shape: Circle | Rectangle): number {
  return shape.area();
}

const circle: Circle = { radius: 5, area() { return Math.PI * this.radius ** 2; } };
const rectangle: Rectangle = { width: 4, height: 6, area() { return this.width * this.height; } };
console.log(calculateArea(circle)); // 78.5398...
console.log(calculateArea(rectangle)); // 24

// 10. Função handleResponse
function handleResponse(response: string | { status: number; data: string }) {
  if (typeof response === "string") {
    console.log(response);
  } else {
    console.log(`Status: ${response.status}, Data: ${response.data}`);
  }
}
handleResponse("Server is running");
handleResponse({ status: 200, data: "OK" });

```