Guia Completo: HTML, CSS e JavaScript para Programação Web

Índice

Introdução às Tecnologias Web {#introdução}

Diferenças Fundamentais

HTML (Estrutura)

- Linguagem de marcação (não é linguagem de programação)
- Define o conteúdo e a estrutura da página
- Usa tags (elementos) para organizar informações
- Exemplo: cabeçalhos, parágrafos, imagens, links

CSS (Apresentação)

- Linguagem de estilo (não é linguagem de programação)
- Define a aparência visual dos elementos HTML
- Controla cores, fontes, layouts, animações
- Separa conteúdo (HTML) de apresentação

JavaScript (Comportamento)

- Linguagem de programação completa
- Adiciona interatividade e dinamismo
- Manipula HTML e CSS em tempo real
- Responde a eventos do usuário

Analogia Útil

- **HTML** = Estrutura de uma casa (paredes, portas, janelas)
- **CSS** = Decoração (cores, móveis, estilo)
- **JavaScript** = Funcionalidades (luz que acende, portas que abrem)

HTML - HyperText Markup Language {#html}

1. Estrutura Básica

```
html
```

```html
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Título da Página</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <!-- Conteúdo visível da página -->
    <h1>Meu Site</h1>

    <script src="script.js"></script>
</body>
</html>
```

Explicação dos Elementos:

- `<!DOCTYPE html>`: Declara que é um documento HTML5
- `<html>`: Elemento raiz, contém todo o documento
- `<head>`: Metadados, não visíveis ao usuário
- `<meta charset="UTF-8">`: Define codificação de caracteres (acentos, ç)
- `<meta name="viewport">`: Responsividade para dispositivos móveis
- `<title>`: Título que aparece na aba do navegador
- `<body>`: Conteúdo visível da página

2. Tags Essenciais

Cabeçalhos (Headings)

```html
<h1>Título Principal</h1>       <!-- Mais importante -->
<h2>Subtítulo</h2>
<h3>Sub-subtítulo</h3>
<h4>Título de 4° nível</h4>
<h5>Título de 5° nível</h5>
<h6>Título de 6° nível</h6>     <!-- Menos importante -->
```

Importante: Use apenas um `<h1>` por página (SEO e acessibilidade)

Texto

html

```html
<p>Parágrafo de texto normal</p>
<strong>Texto em negrito (importância semântica)</strong>
<b>Texto em negrito (apenas visual)</b>
<em>Texto em itálico (ênfase semântica)</em>
<i>Texto em itálico (apenas visual)</i>
<br>  <!-- Quebra de linha -->
<hr>  <!-- Linha horizontal -->
<span>Elemento inline genérico</span>
```

Links

html

```html
<!-- Link externo -->
<a href="https://www.google.com">Ir para Google</a>

<!-- Link interno (mesma página) -->
<a href="#secao">Ir para seção</a>
<div id="secao">Conteúdo da seção</div>

<!-- Link para email -->
<a href="mailto:email@exemplo.com">Enviar email</a>

<!-- Link abre em nova aba -->
<a href="https://exemplo.com" target="_blank">Abrir em nova aba</a>
```

Imagens

html

```html
<!-- Sintaxe básica -->
<img src="caminho/imagem.jpg" alt="Descrição da imagem">

<!-- Com atributos adicionais -->
<img src="foto.jpg" alt="Foto de perfil" width="300" height="200">

<!-- Imagem como link -->
<a href="https://exemplo.com">
    <img src="logo.png" alt="Logo da empresa">
</a>
```

Atributo `alt`: SEMPRE use! Importante para acessibilidade e SEO

Listas

Lista Não Ordenada (bullets):

```html
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ul>
```

Lista Ordenada (números):

```html
<ol>
    <li>Primeiro item</li>
    <li>Segundo item</li>
    <li>Terceiro item</li>
</ol>
```

Lista de Definições:

```html
```

```html
<dl>
    <dt>HTML</dt>
    <dd>Linguagem de marcação para estrutura de páginas</dd>
    <dt>CSS</dt>
    <dd>Linguagem para estilização visual</dd>
</dl>
```

Tabelas


html

```html
<table>
    <thead>
        <tr>
            <th>Cabeçalho 1</th>
            <th>Cabeçalho 2</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Dado 1</td>
            <td>Dado 2</td>
        </tr>
        <tr>
            <td>Dado 3</td>
            <td>Dado 4</td>
        </tr>
    </tbody>
</table>
```

Atributos úteis:

- `colspan="2"`: Célula ocupa 2 colunas
- `rowspan="2"`: Célula ocupa 2 linhas

3. Formulários (MUITO IMPORTANTE!)


html

```html
<form action="/processar" method="POST">
    <!-- Texto simples -->
    <label for="nome">Nome:</label>
    <input type="text" id="nome" name="nome" required>

    <!-- Email -->
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>

    <!-- Senha -->
    <label for="senha">Senha:</label>
    <input type="password" id="senha" name="senha">

    <!-- Número -->
    <label for="idade">Idade:</label>
    <input type="number" id="idade" name="idade" min="0" max="120">

    <!-- Data -->
    <input type="date" name="nascimento">

    <!-- Radio buttons (escolha única) -->
    <input type="radio" id="masculino" name="genero" value="M">
    <label for="masculino">Masculino</label>

    <input type="radio" id="feminino" name="genero" value="F">
    <label for="feminino">Feminino</label>

    <!-- Checkboxes (múltiplas escolhas) -->
    <input type="checkbox" id="html" name="linguagens" value="html">
    <label for="html">HTML</label>

    <input type="checkbox" id="css" name="linguagens" value="css">
    <label for="css">CSS</label>

    <!-- Select (dropdown) -->
    <label for="estado">Estado:</label>
    <select id="estado" name="estado">
        <option value="">Selecione...</option>
        <option value="SP">São Paulo</option>
        <option value="RJ">Rio de Janeiro</option>
        <option value="MG">Minas Gerais</option>
    </select>

    <!-- Textarea (texto longo) -->
```

```
    <label for="mensagem">Mensagem:</label>
    <textarea id="mensagem" name="mensagem" rows="5" cols="30"></textarea>

    <!-- Botões -->
    <button type="submit">Enviar</button>
    <button type="reset">Limpar</button>
    <button type="button">Botão customizado</button>
</form>
```

Atributos importantes de formulário:

- `action`: URL para onde os dados serão enviados
- `method`: GET (dados na URL) ou POST (dados ocultos)
- `required`: Campo obrigatório
- `placeholder`: Texto de exemplo no campo
- `disabled`: Campo desabilitado
- `readonly`: Campo apenas leitura

4. Tags Semânticas HTML5 (IMPORTANTE!)

html

```html
<header>
    <nav>
        <ul>
            <li><a href="#home">Home</a></li>
            <li><a href="#sobre">Sobre</a></li>
        </ul>
    </nav>
</header>

<main>
    <article>
        <h2>Título do Artigo</h2>
        <p>Conteúdo do artigo...</p>
    </article>

    <section>
        <h2>Seção de Produtos</h2>
        <p>Lista de produtos...</p>
    </section>

    <aside>
        <h3>Sidebar</h3>
        <p>Conteúdo relacionado...</p>
    </aside>
</main>

<footer>
    <p>&copy; 2024 Minha Empresa</p>
</footer>
```

Significado das tags semânticas:

- `<header>`: Cabeçalho da página ou seção
- `<nav>`: Menu de navegação
- `<main>`: Conteúdo principal (único na página)
- `<article>`: Conteúdo independente (post, notícia)
- `<section>`: Seção temática do documento
- `<aside>`: Conteúdo relacionado (sidebar)
- `<footer>`: Rodapé da página ou seção

Por que usar?

- Melhora acessibilidade
- Melhora SEO
- Código mais legível e organizado

5. Elementos de Mídia

```html
<!-- Áudio -->
<audio controls>
    <source src="musica.mp3" type="audio/mpeg">
    <source src="musica.ogg" type="audio/ogg">
    Seu navegador não suporta áudio.
</audio>

<!-- Vídeo -->
<video width="640" height="360" controls>
    <source src="video.mp4" type="video/mp4">
    <source src="video.webm" type="video/webm">
    Seu navegador não suporta vídeo.
</video>

<!-- YouTube embed -->
<iframe width="560" height="315"
    src="https://www.youtube.com/embed/VIDEO_ID">
</iframe>
```

6. Comentários e Caracteres Especiais

```html
<!-- Este é um comentário em HTML -->
<!-- Comentários não aparecem na página -->

<!-- Caracteres especiais -->
&lt;     <!-- < (menor que) -->
&gt;     <!-- > (maior que) -->
&amp;    <!-- & (e comercial) -->
&quot;   <!-- " (aspas) -->
&copy;   <!-- © (copyright) -->
    <!-- espaço não quebrável -->
```

7. Atributos Globais (usam em qualquer tag)

```html
<!-- id: identificador único -->
<div id="container"></div>

<!-- class: classe CSS (pode repetir) -->
<p class="texto destaque"></p>

<!-- style: CSS inline (não recomendado) -->
<p style="color: red;">Texto vermelho</p>

<!-- title: tooltip ao passar o mouse -->
<abbr title="HyperText Markup Language">HTML</abbr>

<!-- data-*: atributos personalizados -->
<div data-user-id="123" data-role="admin"></div>
```

CSS - Cascading Style Sheets {#css}

1. Formas de Adicionar CSS

CSS Inline (dentro da tag)

```html
<p style="color: blue; font-size: 16px;">Texto azul</p>
```

Prioridade: Mais alta, mas NÃO recomendado

CSS Interno (dentro do <head>)

```html
```

```
<head>
    <style>
        p {
            color: blue;
            font-size: 16px;
        }
    </style>
</head>
```

Uso: Pequenos projetos ou estilos específicos de uma página

CSS Externo (arquivo separado) - RECOMENDADO


html

```
<head>
    <link rel="stylesheet" href="estilo.css">
</head>
```


css

```
/* arquivo estilo.css */
p {
    color: blue;
    font-size: 16px;
}
```

Vantagens: Reutilização, manutenção, performance

2. Sintaxe Básica


css

```
seletor {
    propriedade: valor;
    outra-propriedade: valor;
}
```

Exemplo:


css

```css
h1 {
    color: red;
    font-size: 32px;
    text-align: center;
}
```

3. Seletores (MUITO IMPORTANTE!)

Seletores Básicos


css

```css
/* Seletor de tag (elemento) */
p {
    color: black;
}

/* Seletor de classe (começa com .) */
.destaque {
    background-color: yellow;
}

/* Seletor de ID (começa com #) */
#cabecalho {
    height: 100px;
}

/* Seletor universal (todos os elementos) */
* {
    margin: 0;
    padding: 0;
}
```

HTML correspondente:


html

```
<p class="destaque">Parágrafo com classe</p>
<div id="cabecalho">Div com ID</div>
```

Seletores Combinados

css

```
/* Múltiplos seletores (mesmas regras) */
h1, h2, h3 {
    font-family: Arial, sans-serif;
}

/* Descendente (espaço) - qualquer nível */
div p {
    color: blue;  /* todos os <p> dentro de <div> */
}

/* Filho direto (>) - apenas primeiro nível */
div > p {
    color: red;  /* apenas <p> filhos diretos de <div> */
}

/* Irmão adjacente (+) - imediatamente depois */
h1 + p {
    font-weight: bold;  /* <p> logo após <h1> */
}

/* Irmãos gerais (~) - todos depois */
h1 ~ p {
    color: gray;  /* todos os <p> após <h1> */
}
```

Seletores de Atributo

css

```css
/* Tem o atributo */
input[required] {
    border: 2px solid red;
}

/* Atributo com valor específico */
input[type="text"] {
    background-color: white;
}

/* Atributo começa com valor */
a[href^="https"] {
    color: green;
}

/* Atributo termina com valor */
img[src$=".png"] {
    border: 1px solid black;
}

/* Atributo contém valor */
a[href*="google"] {
    color: blue;
}
```

Pseudo-classes (estados)

css

```css
/* Link não visitado */
a:link {
    color: blue;
}

/* Link visitado */
a:visited {
    color: purple;
}

/* Mouse sobre o elemento */
a:hover {
    color: red;
    text-decoration: underline;
}

/* Elemento ativo (sendo clicado) */
a:active {
    color: orange;
}

/* Elemento com foco (input selecionado) */
input:focus {
    border: 2px solid blue;
    outline: none;
}

/* Primeiro filho */
li:first-child {
    font-weight: bold;
}

/* Último filho */
li:last-child {
    color: red;
}

/* Enésimo filho */
li:nth-child(2) {
    background-color: yellow;
}

/* Filhos pares */
tr:nth-child(even) {
```

```
    background-color: #f0f0f0;
}

/* Filhos ímpares */
tr:nth-child(odd) {
    background-color: white;
}
```

Pseudo-elementos (partes do elemento)

```css
/* Primeira letra */
p::first-letter {
    font-size: 32px;
    font-weight: bold;
}

/* Primeira linha */
p::first-line {
    font-weight: bold;
}

/* Antes do conteúdo */
h1::before {
    content: "→ ";
    color: red;
}

/* Depois do conteúdo */
h1::after {
    content: " ←";
    color: red;
}

/* Texto selecionado */
::selection {
    background-color: yellow;
    color: black;
}
```

4. Especificidade e Cascata

Ordem de prioridade (maior para menor):

1. `!important` (evite usar!)
2. Inline style (style="...")
3. ID (#id)
4. Classe (.classe), atributo, pseudo-classe
5. Elemento (tag)

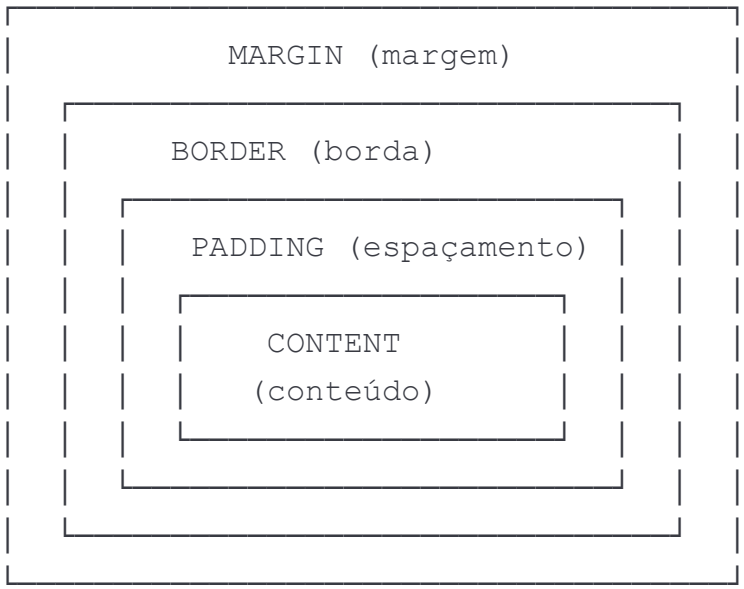
Cálculo de especificidade:

```css
/* 0-0-1 (1 elemento) */
p { color: black; }

/* 0-1-0 (1 classe) */
.texto { color: blue; }

/* 1-0-0 (1 ID) */
#principal { color: red; }

/* 0-1-1 (1 classe + 1 elemento) */
p.texto { color: green; }

/* 1-1-1 (1 ID + 1 classe + 1 elemento) */
div#container p.destaque { color: purple; }
```

Cascata: Se mesma especificidade, última regra ganha

5. Box Model (CONCEITO FUNDAMENTAL!)

Todo elemento HTML é uma caixa retangular composta por:


MARGIN (margem)
BORDER (borda)
PADDING (espaçamento)
CONTENT
(conteúdo)

css

```css
div {
    /* Conteúdo */
    width: 300px;
    height: 200px;

    /* Padding (espaço interno) */
    padding: 20px;
    /* ou específico */
    padding-top: 10px;
    padding-right: 15px;
    padding-bottom: 10px;
    padding-left: 15px;
    /* shorthand */
    padding: 10px 15px 10px 15px; /* top right bottom left */
    padding: 10px 15px; /* top/bottom left/right */

    /* Border (borda) */
    border: 2px solid black;
    /* ou específico */
    border-width: 2px;
    border-style: solid; /* solid, dashed, dotted, double */
    border-color: black;
    border-radius: 10px; /* bordas arredondadas */

    /* Margin (espaço externo) */
    margin: 20px;
    /* centralizar horizontalmente */
    margin: 0 auto;
}

/* Box-sizing: altera cálculo da largura */
* {
    box-sizing: border-box; /* largura inclui padding e border */
}
```

6. Propriedades de Texto

CSS

```css
p {
    /* Cor */
    color: #333;
    color: rgb(51, 51, 51);
    color: rgba(51, 51, 51, 0.8); /* com transparência */

    /* Fonte */
    font-family: Arial, Helvetica, sans-serif;
    font-size: 16px;
    font-weight: bold; /* normal, bold, 100-900 */
    font-style: italic; /* normal, italic */

    /* Alinhamento */
    text-align: left; /* left, right, center, justify */

    /* Decoração */
    text-decoration: underline; /* none, underline, line-through */

    /* Transformação */
    text-transform: uppercase; /* uppercase, lowercase, capitalize */

    /* Espaçamento */
    line-height: 1.5; /* espaço entre linhas */
    letter-spacing: 2px; /* espaço entre letras */
    word-spacing: 5px; /* espaço entre palavras */

    /* Sombra */
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
    /* horizontal vertical blur cor */
}
```

7. Cores e Backgrounds

css

```css
div {
    /* Cor de fundo */
    background-color: #f0f0f0;

    /* Imagem de fundo */
    background-image: url('imagem.jpg');
    background-repeat: no-repeat; /* repeat, repeat-x, repeat-y */
    background-position: center center; /* top, bottom, left, right, center */
    background-size: cover; /* contain, cover, 100px 200px */
    background-attachment: fixed; /* scroll, fixed */

    /* Shorthand */
    background: #f0f0f0 url('imagem.jpg') no-repeat center/cover;

    /* Gradientes */
    background: linear-gradient(to right, red, blue);
    background: linear-gradient(45deg, #ff0000, #00ff00, #0000ff);
    background: radial-gradient(circle, red, yellow);
}
```

8. Display e Posicionamento

Display

CSS

```css
/* Block: ocupa toda largura, quebra linha */
div {
    display: block;
}

/* Inline: ocupa só o necessário, não quebra linha */
span {
    display: inline;
}

/* Inline-block: híbrido (aceita width/height, não quebra linha) */
img {
    display: inline-block;
}

/* None: esconde elemento */
.oculto {
    display: none;
}

/* Flex: layout flexível (IMPORTANTE!) */
.container {
    display: flex;
}

/* Grid: layout em grade */
.grade {
    display: grid;
}
```

Position

CSS

```css
/* Static: padrão, fluxo normal */
div {
    position: static;
}

/* Relative: relativo à posição original */
.relativo {
    position: relative;
    top: 10px;  /* desloca 10px para baixo */
    left: 20px; /* desloca 20px para direita */
}

/* Absolute: relativo ao ancestral posicionado */
.absoluto {
    position: absolute;
    top: 0;
    right: 0; /* canto superior direito do pai */
}

/* Fixed: fixo na janela (scroll não afeta) */
.fixo {
    position: fixed;
    bottom: 20px;
    right: 20px; /* sempre no canto inferior direito */
}

/* Sticky: mix de relative e fixed */
.sticky {
    position: sticky;
    top: 0; /* gruda no topo ao rolar */
}

/* Z-index: controla sobreposição */
.frente {
    position: relative;
    z-index: 10; /* maior = mais à frente */
}
```

9. Flexbox (MUITO IMPORTANTE!)

css

```css
/* Container Flex */
.container {
    display: flex;

    /* Direção dos itens */
    flex-direction: row; /* row, row-reverse, column, column-reverse */

    /* Quebra de linha */
    flex-wrap: wrap; /* nowrap, wrap, wrap-reverse */

    /* Alinhamento horizontal (eixo principal) */
    justify-content: center; /* flex-start, flex-end, center, space-between, spa

    /* Alinhamento vertical (eixo transversal) */
    align-items: center; /* flex-start, flex-end, center, stretch, baseline */

    /* Alinhamento de múltiplas linhas */
    align-content: center;

    /* Gap entre itens */
    gap: 20px; /* espaço entre itens */
}

/* Itens Flex */
.item {
    /* Crescimento */
    flex-grow: 1; /* proporção de crescimento */

    /* Encolhimento */
    flex-shrink: 1; /* proporção de encolhimento */

    /* Tamanho base */
    flex-basis: 200px; /* tamanho inicial */

    /* Shorthand */
    flex: 1 1 200px; /* grow shrink basis */

    /* Alinhamento individual */
    align-self: flex-end; /* sobrescreve align-items */

    /* Ordem */
```

```
    order: 2; /* ordem de exibição */
}
```

Exemplo prático:

```css
.navbar {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 20px;
}

.cards {
    display: flex;
    flex-wrap: wrap;
    gap: 20px;
}

.card {
    flex: 1 1 300px; /* cresce, encolhe, mínimo 300px */
}
```

10. Grid Layout

```css
```

```css
/* Container Grid */
.container {
    display: grid;

    /* Colunas */
    grid-template-columns: 200px 1fr 200px; /* 3 colunas */
    grid-template-columns: repeat(3, 1fr); /* 3 colunas iguais */
    grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); /* responsivo

    /* Linhas */
    grid-template-rows: 100px auto 100px;

    /* Gap */
    gap: 20px; /* espaço entre células */
    column-gap: 20px;
    row-gap: 10px;

    /* Áreas nomeadas */
    grid-template-areas:
        "header header header"
        "sidebar main main"
        "footer footer footer";
}

/* Itens Grid */
.item {
    /* Posicionamento por linha */
    grid-column: 1 / 3; /* da coluna 1 até 3 */
    grid-row: 1 / 2;

    /* Span (quantas células ocupa) */
    grid-column: span 2; /* ocupa 2 colunas */

    /* Áreas nomeadas */
    grid-area: header;
}

/* Exemplo prático */
.header { grid-area: header; }
.sidebar { grid-area: sidebar; }
.main { grid-area: main; }
.footer { grid-area: footer; }
```

11. Transições e Animações

Transições (mudança suave)

```css
button {
    background-color: blue;
    transition: background-color 0.3s ease;
    /* propriedade duração timing-function */
}

button:hover {
    background-color: red;
}

/* Múltiplas propriedades */
.box {
    transition: width 0.3s, height 0.3s, background-color 0.5s;
}

/* Todas propriedades */
.box {
    transition: all 0.3s ease-in-out;
}
```

Timing functions:

- `ease`: lento-rápido-lento (padrão)
- `linear`: velocidade constante
- `ease-in`: começa lento
- `ease-out`: termina lento
- `ease-in-out`: começa e termina lento

Animações (sequências complexas)

```css
```

```css
/* Definir animação */
@keyframes deslizar {
    0% {
        transform: translateX(0);
        opacity: 0;
    }
    50% {
        opacity: 1;
    }
    100% {
        transform: translateX(100px);
        opacity: 0;
    }
}

/* Aplicar animação */
.elemento {
    animation: deslizar 2s ease-in-out infinite;
    /* nome duração timing repetição */
}

/* Propriedades detalhadas */
.elemento {
    animation-name: deslizar;
    animation-duration: 2s;
    animation-timing-function: ease;
    animation-delay: 0.5s; /* atraso */
    animation-iteration-count: infinite; /* ou número */
    animation-direction: alternate; /* normal, reverse, alternate */
    animation-fill-mode: forwards; /* mantém estado final */
    animation-play-state: paused; /* pausa animação */
}
```

12. Transform (Transformações)

CSS

```css
.elemento {
    /* Translação (mover) */
    transform: translate(50px, 100px); /* x, y */
    transform: translateX(50px);
    transform: translateY(100px);

    /* Rotação */
    transform: rotate(45deg); /* graus */

    /* Escala (redimensionar) */
    transform: scale(1.5); /* 150% */
    transform: scale(2, 0.5); /* x, y */

    /* Inclinação */
    transform: skew(20deg, 10deg);

    /* Múltiplas transformações */
    transform: translate(50px, 100px) rotate(45deg) scale(1.5);

    /* Origem da transformação */
    transform-origin: center; /* top, bottom, left, right, center */
    transform-origin: 50% 50%; /* x y */
}

/* Exemplo prático: Card com hover */
.card {
    transition: transform 0.3s;
}

.card:hover {
    transform: translateY(-10px) scale(1.05);
}
```

13. Responsividade - Media Queries (ESSENCIAL!)

CSS

```css
/* Mobile First (começa do menor) */
.container {
    width: 100%;
    padding: 10px;
}

/* Tablets e acima (768px+) */
@media (min-width: 768px) {
    .container {
        width: 750px;
        padding: 20px;
    }
}

/* Desktop e acima (1024px+) */
@media (min-width: 1024px) {
    .container {
        width: 1000px;
        padding: 30px;
    }
}

/* Desktop grande (1200px+) */
@media (min-width: 1200px) {
    .container {
        width: 1140px;
    }
}

/* Orientação */
@media (orientation: landscape) {
    /* Modo paisagem */
}

@media (orientation: portrait) {
    /* Modo retrato */
}

/* Múltiplas condições */
@media (min-width: 768px) and (max-width: 1024px) {
    /* Apenas para tablets */
}

/* Impressão */
```

```css
@media print {
    .no-print {
        display: none;
    }
}
```

14. Outras Propriedades Úteis

CSS

```css
/* Visibilidade */
.elemento {
    visibility: hidden; /* oculta mas ocupa espaço */
    opacity: 0.5; /* transparência 0-1 */
}

/* Overflow (conteúdo excedente) */
.caixa {
    overflow: hidden; /* esconde */
    overflow: scroll; /* sempre mostra scroll */
    overflow: auto; /* scroll se necessário */
    overflow-x: hidden; /* apenas horizontal */
    overflow-y: scroll; /* apenas vertical */
}

/* Cursor */
.link {
    cursor: pointer; /* mãozinha */
    cursor: not-allowed; /* proibido */
    cursor: help; /* interrogação */
}

/* Lista */
ul {
    list-style-type: none; /* remove marcadores */
    list-style-type: circle; /* disc, square, decimal */
    list-style-position: inside; /* outside, inside */
    list-style-image: url('bullet.png');
}

/* Sombras */
.box {
    box-shadow: 5px 5px 10px rgba(0, 0, 0, 0.3);
    /* horizontal vertical blur spread cor */
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1); /* sombra sutil */
}

/* Filtros */
img {
    filter: grayscale(100%); /* preto e branco */
    filter: blur(5px); /* desfoque */
    filter: brightness(150%); /* brilho */
    filter: contrast(200%); /* contraste */
```

```
    filter: sepia(100%); /* sépia */
}
```

JavaScript {#javascript}

1. Introdução e Conceitos Básicos

JavaScript é uma **linguagem de programação** de alto nível, interpretada, orientada a objetos e com tipagem dinâmica.

Formas de Adicionar JavaScript

JavaScript Inline:

html

```
<button onclick="alert('Clicou!')">Clique</button>
```

JavaScript Interno:

html

```
<script>
    console.log('Olá, mundo!');
</script>
```

JavaScript Externo (RECOMENDADO):

html

```
<!-- No final do body -->
<script src="script.js"></script>
```

2. Variáveis e Tipos de Dados

Declaração de Variáveis

javascript

```javascript
// var (escopo de função, evite usar)
var nome = 'João';

// let (escopo de bloco, pode reatribuir)
let idade = 25;
idade = 26; // OK

// const (escopo de bloco, NÃO pode reatribuir)
const PI = 3.14159;
// PI = 3; // ERRO!

// Múltiplas declarações
let x = 10, y = 20, z = 30;
```

Diferenças importantes:

- `let` e `const` têm escopo de bloco `{}`
- `var` tem escopo de função e sofre "hoisting"
- Use `const` por padrão, `let` quando precisar reatribuir

Tipos de Dados Primitivos

javascript

```javascript
// String (texto)
let nome = "Maria";
let sobrenome = 'Silva';
let frase = `Olá, ${nome}!`; // Template string

// Number (número)
let inteiro = 42;
let decimal = 3.14;
let negativo = -10;
let infinito = Infinity;
let naoNumero = NaN; // Not a Number

// Boolean (verdadeiro/falso)
let ativo = true;
let desativado = false;

// Undefined (não definido)
let indefinido;
console.log(indefinido); // undefined

// Null (vazio intencional)
let vazio = null;

// Symbol (único, ES6)
let simbolo = Symbol('descrição');

// BigInt (números muito grandes, ES2020)
let grande = 9007199254740991n;
```

Verificar Tipo

javascript

```javascript
typeof "texto";       // "string"
typeof 42;            // "number"
typeof true;          // "boolean"
typeof undefined;     // "undefined"
typeof null;          // "object" (bug histórico!)
typeof {};            // "object"
typeof [];            // "object"
typeof function(){}; // "function"
```

3. Operadores

Aritméticos

javascript

```javascript
let a = 10, b = 3;

a + b;  // 13 (adição)
a - b;  // 7 (subtração)
a * b;  // 30 (multiplicação)
a / b;  // 3.333... (divisão)
a % b;  // 1 (resto da divisão - módulo)
a ** b; // 1000 (potenciação)

// Incremento e Decremento
let x = 5;
x++;    // x = 6 (pós-incremento)
++x;    // x = 7 (pré-incremento)
x--;    // x = 6 (pós-decremento)
--x;    // x = 5 (pré-decremento)
```

Atribuição

javascript

```javascript
let x = 10;

x += 5;  // x = x + 5  (x = 15)
x -= 3;  // x = x - 3  (x = 12)
x *= 2;  // x = x * 2  (x = 24)
x /= 4;  // x = x / 4  (x = 6)
x %= 4;  // x = x % 4  (x = 2)
```

Comparação

javascript

```javascript
// Igualdade (compara valor, converte tipos)
5 == "5";    // true
5 == 5;      // true

// Igualdade estrita (compara valor E tipo)
5 === "5";   // false
5 === 5;     // true

// Diferença
5 != "5";    // false
5 !== "5";   // true (estrita)

// Relacionais
10 > 5;      // true
10 < 5;      // false
10 >= 10;    // true
10 <= 5;     // false
```

IMPORTANTE: Sempre use === e !== (igualdade estrita)!

Lógicos

javascript

```javascript
// AND (&&) - ambos verdadeiros
true && true;   // true
true && false;  // false

// OR (||) - pelo menos um verdadeiro
true || false;  // true
false || false; // false

// NOT (!) - inverte
!true;          // false
!false;         // true

// Exemplos práticos
let idade = 20;
let temCarteira = true;

if (idade >= 18 && temCarteira) {
    console.log('Pode dirigir');
}

// Short-circuit
let nome = usuario || 'Anônimo'; // se usuario for falsy, usa 'Anônimo'
```

Ternário (if resumido)

javascript

```javascript
// condição ? valorSeVerdadeiro : valorSeFalso
let idade = 20;
let status = idade >= 18 ? 'Maior de idade' : 'Menor de idade';

// Aninhado (evite complexidade)
let nota = 85;
let conceito = nota >= 90 ? 'A' : nota >= 80 ? 'B' : nota >= 70 ? 'C' : 'D';
```

4. Estruturas de Controle

If / Else

javascript

```javascript
let nota = 75;

if (nota >= 90) {
    console.log('Excelente');
} else if (nota >= 70) {
    console.log('Bom');
} else if (nota >= 50) {
    console.log('Regular');
} else {
    console.log('Insuficiente');
}

// If sem else
if (idade >= 18) {
    console.log('Maior de idade');
}

// Valores falsy (considerados false)
// false, 0, "", null, undefined, NaN
if (0) {
    // Não executa
}

if ("texto") {
    // Executa (string não vazia é truthy)
}
```

Switch

javascript

```javascript
let dia = 3;
let nomeDia;

switch (dia) {
    case 1:
        nomeDia = 'Segunda';
        break;
    case 2:
        nomeDia = 'Terça';
        break;
    case 3:
        nomeDia = 'Quarta';
        break;
    case 4:
        nomeDia = 'Quinta';
        break;
    case 5:
        nomeDia = 'Sexta';
        break;
    case 6:
    case 7:
        nomeDia = 'Fim de semana';
        break;
    default:
        nomeDia = 'Dia inválido';
}

console.log(nomeDia); // "Quarta"
```

IMPORTANTE: Não esqueça o `break`! Sem ele, executa os próximos cases.

5. Loops (Laços de Repetição)

For

javascript

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

// Contagem regressiva
for (let i = 10; i > 0; i--) {
    console.log(i);
}

// Múltiplas variáveis
for (let i = 0, j = 10; i < j; i++, j--) {
    console.log(i, j);
}

// Loop infinito (cuidado!)
for (;;) {
    // Executa para sempre
    break; // Use break para sair
}
```

While

javascript

```javascript
// While (testa antes de executar)
let i = 0;
while (i < 5) {
    console.log(i);
    i++;
}

// Exemplo prático
let senha;
while (senha !== '1234') {
    senha = prompt('Digite a senha:');
}
```

Do...While

javascript

```javascript
// Executa pelo menos uma vez
let numero;
do {
    numero = prompt('Digite um número maior que 10:');
} while (numero <= 10);
```

For...of (arrays)

javascript

```javascript
let frutas = ['maçã', 'banana', 'laranja'];

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

For...in (objetos e índices)

javascript

```javascript
// Para objetos
let pessoa = { nome: 'João', idade: 30, cidade: 'SP' };

for (let chave in pessoa) {
    console.log(chave + ': ' + pessoa[chave]);
}
// nome: João
// idade: 30
// cidade: SP

// Para arrays (retorna índices, não valores!)
let cores = ['vermelho', 'verde', 'azul'];
for (let indice in cores) {
    console.log(indice, cores[indice]);
}
```

Break e Continue

javascript

```javascript
// Break: sai do loop
for (let i = 0; i < 10; i++) {
    if (i === 5) {
        break; // Para no 5
    }
    console.log(i); // 0, 1, 2, 3, 4
}

// Continue: pula para próxima iteração
for (let i = 0; i < 10; i++) {
    if (i % 2 === 0) {
        continue; // Pula números pares
    }
    console.log(i); // 1, 3, 5, 7, 9
}
```

6. Funções (MUITO IMPORTANTE!)

Declaração de Função

javascript

```javascript
// Função nomeada
function saudacao(nome) {
    return 'Olá, ' + nome + '!';
}

let mensagem = saudacao('Maria'); // "Olá, Maria!"

// Função sem retorno (retorna undefined)
function imprimirMensagem(texto) {
    console.log(texto);
}

// Múltiplos parâmetros
function somar(a, b) {
    return a + b;
}

// Parâmetros padrão
function cumprimentar(nome = 'Visitante') {
    return 'Olá, ' + nome;
}

cumprimentar();         // "Olá, Visitante"
cumprimentar('João');   // "Olá, João"
```

Expressão de Função

javascript

```javascript
// Função anônima atribuída a variável
const multiplicar = function(a, b) {
    return a * b;
};

let resultado = multiplicar(5, 3); // 15
```

Arrow Functions (ES6)

javascript

```javascript
// Sintaxe básica
const somar = (a, b) => {
    return a + b;
};

// Retorno implícito (uma linha)
const somar = (a, b) => a + b;

// Um parâmetro (parênteses opcionais)
const dobro = x => x * 2;

// Sem parâmetros
const saudacao = () => 'Olá!';

// Retornar objeto (use parênteses)
const criarPessoa = (nome, idade) => ({ nome: nome, idade: idade });

// Arrow function não tem 'this' próprio
const pessoa = {
    nome: 'João',
    idade: 30,
    apresentar: function() {
        // Function tradicional tem 'this'
        setTimeout(() => {
            // Arrow function herda 'this' do contexto
            console.log(`Meu nome é ${this.nome}`);
        }, 1000);
    }
};
```

Escopo de Variáveis

javascript

```javascript
let global = 'Variável global';

function exemplo() {
    let local = 'Variável local';
    console.log(global); // Acessa global
    console.log(local);  // Acessa local
}

exemplo();
// console.log(local); // ERRO! local não existe fora da função

// Escopo de bloco
if (true) {
    let bloqueada = 'Só existe aqui';
    var vazada = 'Vaza do bloco';
}

// console.log(bloqueada); // ERRO!
console.log(vazada); // Funciona (var não respeita bloco)
```

Closures (função dentro de função)

```javascript
function contador() {
    let count = 0; // Variável privada

    return function() {
        count++;
        return count;
    };
}

const incrementar = contador();
console.log(incrementar()); // 1
console.log(incrementar()); // 2
console.log(incrementar()); // 3
```

7. Arrays (ESSENCIAL!)

Criação e Acesso

javascript

```javascript
// Criação
let frutas = ['maçã', 'banana', 'laranja'];
let numeros = [1, 2, 3, 4, 5];
let misto = [1, 'texto', true, null, { nome: 'João' }];
let vazio = [];

// Acesso (índice começa em 0)
console.log(frutas[0]);  // "maçã"
console.log(frutas[1]);  // "banana"
console.log(frutas[2]);  // "laranja"

// Modificar
frutas[1] = 'morango';
console.log(frutas); // ['maçã', 'morango', 'laranja']

// Tamanho
console.log(frutas.length); // 3

// Último elemento
let ultimo = frutas[frutas.length - 1];
```

Métodos de Array (MUITO COBRADOS!)

javascript

```javascript
let numeros = [1, 2, 3, 4, 5];

// push: adiciona no final
numeros.push(6);
console.log(numeros); // [1, 2, 3, 4, 5, 6]

// pop: remove do final
let removido = numeros.pop();
console.log(removido);  // 6
console.log(numeros);   // [1, 2, 3, 4, 5]

// unshift: adiciona no início
numeros.unshift(0);
console.log(numeros); // [0, 1, 2, 3, 4, 5]

// shift: remove do início
let primeiro = numeros.shift();
console.log(primeiro); // 0
console.log(numeros);  // [1, 2, 3, 4, 5]

// splice: adiciona/remove em qualquer posição
// splice(índice, quantos_remover, elementos_adicionar...)
let cores = ['vermelho', 'verde', 'azul'];
cores.splice(1, 1, 'amarelo', 'roxo'); // Remove 1 no índice 1, adiciona 2
console.log(cores); // ['vermelho', 'amarelo', 'roxo', 'azul']

// slice: copia parte do array (não modifica original)
let numeros = [1, 2, 3, 4, 5];
let parte = numeros.slice(1, 4); // do índice 1 até 4 (não inclui 4)
console.log(parte);   // [2, 3, 4]
console.log(numeros); // [1, 2, 3, 4, 5] (original intacto)

// concat: junta arrays
let arr1 = [1, 2];
let arr2 = [3, 4];
let junto = arr1.concat(arr2);
console.log(junto); // [1, 2, 3, 4]

// join: transforma em string
let palavras = ['Olá', 'mundo'];
let frase = palavras.join(' ');
console.log(frase); // "Olá mundo"

// indexOf: encontra índice
```

```javascript
let frutas = ['maçã', 'banana', 'laranja'];
let indice = frutas.indexOf('banana');
console.log(indice); // 1

// includes: verifica se contém
console.log(frutas.includes('banana')); // true
console.log(frutas.includes('uva'));    // false

// reverse: inverte ordem (modifica original)
numeros.reverse();
console.log(numeros); // [5, 4, 3, 2, 1]

// sort: ordena (modifica original)
let nomes = ['Carlos', 'Ana', 'Bruno'];
nomes.sort();
console.log(nomes); // ['Ana', 'Bruno', 'Carlos']

// sort com números (CUIDADO!)
let nums = [10, 5, 40, 25, 1000, 1];
nums.sort(); // ['1', '10', '1000', '25', '40', '5'] ERRADO!
nums.sort((a, b) => a - b); // [1, 5, 10, 25, 40, 1000] CORRETO!
```

Métodos Avançados (Higher-Order Functions)

javascript

```javascript
let numeros = [1, 2, 3, 4, 5];

// forEach: executa função para cada elemento
numeros.forEach(function(numero, indice) {
    console.log(`Índice ${indice}: ${numero}`);
});

// Arrow function
numeros.forEach((num) => console.log(num * 2));

// map: cria novo array transformado
let dobrados = numeros.map(num => num * 2);
console.log(dobrados); // [2, 4, 6, 8, 10]

// filter: cria novo array com elementos que passam no teste
let pares = numeros.filter(num => num % 2 === 0);
console.log(pares); // [2, 4]

// find: retorna primeiro elemento que passa no teste
let encontrado = numeros.find(num => num > 3);
console.log(encontrado); // 4

// findIndex: retorna índice do primeiro que passa no teste
let indice = numeros.findIndex(num => num > 3);
console.log(indice); // 3

// some: verifica se ALGUM elemento passa no teste
let temMaiorQue3 = numeros.some(num => num > 3);
console.log(temMaiorQue3); // true

// every: verifica se TODOS elementos passam no teste
let todosMaioresQue0 = numeros.every(num => num > 0);
console.log(todosMaioresQue0); // true

// reduce: reduz array a um único valor
let soma = numeros.reduce((acumulador, atual) => acumulador + atual, 0);
console.log(soma); // 15

// Exemplo: encontrar maior valor
let maior = numeros.reduce((max, num) => num > max ? num : max);
console.log(maior); // 5
```

8. Objetos (ESSENCIAL!)

Criação e Acesso

javascript

```javascript
// Objeto literal
let pessoa = {
    nome: 'João',
    idade: 30,
    cidade: 'São Paulo',
    ativo: true
};

// Acesso com ponto
console.log(pessoa.nome);  // "João"

// Acesso com colchetes
console.log(pessoa['idade']); // 30

// Adicionar propriedade
pessoa.profissao = 'Desenvolvedor';

// Modificar
pessoa.idade = 31;

// Deletar
delete pessoa.ativo;

// Verificar se propriedade existe
console.log('nome' in pessoa);  // true
console.log(pessoa.hasOwnProperty('nome')); // true
```

Métodos (funções dentro de objetos)

javascript

```javascript
let calculadora = {
    valor: 0,

    somar: function(n) {
        this.valor += n;
        return this; // permite encadear métodos
    },

    subtrair: function(n) {
        this.valor -= n;
        return this;
    },

    // Sintaxe ES6 (sem ': function')
    multiplicar(n) {
        this.valor *= n;
        return this;
    },

    resultado() {
        return this.valor;
    }
};

// Encadeamento
calculadora.somar(10).multiplicar(2).subtrair(5);
console.log(calculadora.resultado()); // 15
```

This (contexto)

javascript

```javascript
let pessoa = {
    nome: 'Maria',
    apresentar: function() {
        console.log('Meu nome é ' + this.nome);
    }
};

pessoa.apresentar(); // "Meu nome é Maria"

// Cuidado! This muda conforme contexto
let funcao = pessoa.apresentar;
funcao(); // "Meu nome é undefined" (this não é mais pessoa)
```

Desestruturação (ES6)

javascript

```javascript
let pessoa = { nome: 'João', idade: 30, cidade: 'SP' };

// Extrair propriedades
let { nome, idade } = pessoa;
console.log(nome);  // "João"
console.log(idade); // 30

// Renomear
let { nome: nomeCompleto, idade: anos } = pessoa;
console.log(nomeCompleto); // "João"

// Valor padrão
let { profissao = 'Desconhecida' } = pessoa;
console.log(profissao); // "Desconhecida"

// Arrays
let cores = ['vermelho', 'verde', 'azul'];
let [primeira, segunda] = cores;
console.log(primeira);  // "vermelho"
console.log(segunda);   // "verde"
```

Métodos Úteis de Object

javascript

```javascript
let pessoa = { nome: 'João', idade: 30 };

// Object.keys: retorna array de chaves
let chaves = Object.keys(pessoa);
console.log(chaves); // ['nome', 'idade']

// Object.values: retorna array de valores
let valores = Object.values(pessoa);
console.log(valores); // ['João', 30]

// Object.entries: retorna array de [chave, valor]
let entradas = Object.entries(pessoa);
console.log(entradas); // [['nome', 'João'], ['idade', 30]]

// Iterar objeto
Object.entries(pessoa).forEach(([chave, valor]) => {
    console.log(`${chave}: ${valor}`);
});

// Object.assign: copiar/mesclar objetos
let copia = Object.assign({}, pessoa);
let mesclado = Object.assign({}, pessoa, { cidade: 'SP' });
console.log(mesclado); // { nome: 'João', idade: 30, cidade: 'SP' }

// Spread operator (...) - ES6
let copia2 = { ...pessoa };
let mesclado2 = { ...pessoa, cidade: 'SP', idade: 31 };
```

9. Strings (Métodos Importantes)

javascript

```javascript
let texto = 'JavaScript é incrível';

// Tamanho
console.log(texto.length); // 21

// Acessar caractere
console.log(texto[0]);           // "J"
console.log(texto.charAt(0));    // "J"

// Maiúsculas/Minúsculas
console.log(texto.toUpperCase()); // "JAVASCRIPT É INCRÍVEL"
console.log(texto.toLowerCase()); // "javascript é incrível"

// Busca
console.log(texto.indexOf('Script'));     // 4
console.log(texto.lastIndexOf('i'));      // 19
console.log(texto.includes('Java'));      // true
console.log(texto.startsWith('Java'));    // true
console.log(texto.endsWith('vel'));       // true

// Extração
console.log(texto.slice(0, 10));          // "JavaScript"
console.log(texto.slice(-8));             // "incrível"
console.log(texto.substring(0, 10));      // "JavaScript"
console.log(texto.substr(0, 10));         // "JavaScript" (deprecated)

// Substituição
console.log(texto.replace('incrível', 'fantástico'));
console.log(texto.replaceAll('i', 'I'));  // Substitui todos

// Split: transforma em array
let palavras = texto.split(' ');
console.log(palavras); // ['JavaScript', 'é', 'incrível']

// Trim: remove espaços das pontas
let comEspacos = '  texto  ';
console.log(comEspacos.trim());      // "texto"
console.log(comEspacos.trimStart()); // "texto  "
console.log(comEspacos.trimEnd());   // "  texto"

// Repetir
console.log('ha'.repeat(3)); // "hahaha"

// Template strings (IMPORTANTE!)
```

```javascript
let nome = 'Maria';
let idade = 25;
let mensagem = `Olá, meu nome é ${nome} e tenho ${idade} anos.`;
console.log(mensagem);

// Multilinhas
let html = `
    <div>
        <h1>${nome}</h1>
        <p>Idade: ${idade}</p>
    </div>
`;
```

10. DOM - Document Object Model (CRUCIAL!)

Selecionar Elementos

javascript

```javascript
// Por ID (retorna 1 elemento ou null)
let titulo = document.getElementById('titulo');

// Por classe (retorna HTMLCollection)
let itens = document.getElementsByClassName('item');

// Por tag (retorna HTMLCollection)
let paragrafos = document.getElementsByTagName('p');

// Query Selector (retorna 1 elemento, CSS selector)
let primeiro = document.querySelector('.item');
let botao = document.querySelector('#meuBotao');

// Query Selector All (retorna NodeList, CSS selector)
let todos = document.querySelectorAll('.item');

// Diferença: NodeList tem forEach, HTMLCollection não
todos.forEach(elemento => {
    console.log(elemento);
});
```

Manipular Conteúdo

javascript

```javascript
let elemento = document.getElementById('texto');

// Texto puro (sem HTML)
elemento.textContent = 'Novo texto';
console.log(elemento.textContent);

// HTML (interpreta tags)
elemento.innerHTML = '<strong>Texto em negrito</strong>';
console.log(elemento.innerHTML);

// Valor de inputs
let input = document.getElementById('nome');
input.value = 'João';
console.log(input.value);
```

Manipular Atributos

javascript

```javascript
let imagem = document.getElementById('foto');

// Pegar atributo
let src = imagem.getAttribute('src');

// Definir atributo
imagem.setAttribute('src', 'nova-foto.jpg');
imagem.setAttribute('alt', 'Descrição');

// Remover atributo
imagem.removeAttribute('title');

// Verificar se tem atributo
if (imagem.hasAttribute('src')) {
    console.log('Tem src');
}

// Acesso direto (propriedades comuns)
imagem.src = 'foto.jpg';
imagem.alt = 'Minha foto';

let link = document.getElementById('meuLink');
link.href = 'https://exemplo.com';
```

Manipular Classes CSS

javascript

```javascript
let elemento = document.getElementById('caixa');

// Adicionar classe
elemento.classList.add('ativo');
elemento.classList.add('destaque', 'grande'); // múltiplas

// Remover classe
elemento.classList.remove('ativo');

// Alternar classe (toggle)
elemento.classList.toggle('escondido'); // adiciona se não tem, remove se tem

// Verificar se tem classe
if (elemento.classList.contains('ativo')) {
    console.log('Está ativo');
}

// Substituir classe
elemento.classList.replace('antigo', 'novo');
```

Manipular Estilos (CSS inline)

javascript

```javascript
let caixa = document.getElementById('caixa');

// Definir estilo
caixa.style.backgroundColor = 'blue';
caixa.style.color = 'white';
caixa.style.fontSize = '20px';
caixa.style.display = 'none';

// CSS properties com hífen viram camelCase:
// background-color → backgroundColor
// font-size → fontSize

// Múltiplos estilos
Object.assign(caixa.style, {
    width: '200px',
    height: '200px',
    border: '2px solid black'
});

// Pegar estilo computado (inclui CSS externo)
let estiloAtual = window.getComputedStyle(caixa);
console.log(estiloAtual.backgroundColor);
```

Criar e Remover Elementos

javascript

```javascript
// Criar elemento
let novoParagrafo = document.createElement('p');
novoParagrafo.textContent = 'Texto do parágrafo';
novoParagrafo.classList.add('paragrafo');

// Adicionar ao DOM
let container = document.getElementById('container');
container.appendChild(novoParagrafo); // adiciona no final

// Inserir em posição específica
let primeiroParagrafo = document.querySelector('p');
container.insertBefore(novoParagrafo, primeiroParagrafo); // antes de outro ele

// Métodos modernos de inserção
container.prepend(novoParagrafo);  // início do container
container.append(novoParagrafo);   // final do container
primeiroParagrafo.before(novoParagrafo); // antes do elemento
primeiroParagrafo.after(novoParagrafo);  // depois do elemento

// Remover elemento
novoParagrafo.remove(); // ES6

// Método antigo
container.removeChild(novoParagrafo);

// Substituir
let novoElemento = document.createElement('div');
container.replaceChild(novoElemento, novoParagrafo);

// Clonar elemento
let clone = novoParagrafo.cloneNode(true); // true = clona filhos também
```

Navegar no DOM

javascript

```javascript
let elemento = document.getElementById('item');

// Pai
let pai = elemento.parentElement;
let paiNode = elemento.parentNode; // pode retornar qualquer tipo de node

// Filhos
let filhos = elemento.children; // HTMLCollection (apenas elementos)
let todosNos = elemento.childNodes; // NodeList (inclui texto, comentários)
let primeiroFilho = elemento.firstElementChild;
let ultimoFilho = elemento.lastElementChild;

// Irmãos
let proximo = elemento.nextElementSibling;
let anterior = elemento.previousElementSibling;

// Ancestral específico
let ancestral = elemento.closest('.container'); // sobe até encontrar
```

11. Eventos (MUITO IMPORTANTE!)

Adicionar Event Listeners

javascript

```javascript
// Forma recomendada: addEventListener
let botao = document.getElementById('meuBotao');

botao.addEventListener('click', function() {
    console.log('Botão clicado!');
});

// Arrow function
botao.addEventListener('click', () => {
    console.log('Clicou!');
});

// Função nomeada (para poder remover depois)
function handleClick() {
    console.log('Clicado!');
}
botao.addEventListener('click', handleClick);

// Remover listener
botao.removeEventListener('click', handleClick);

// Múltiplos listeners no mesmo evento
botao.addEventListener('click', funcao1);
botao.addEventListener('click', funcao2); // ambos executam
```

Tipos de Eventos Comuns

javascript

```javascript
// Mouse
elemento.addEventListener('click', callback);        // clique
elemento.addEventListener('dblclick', callback);     // duplo clique
elemento.addEventListener('mousedown', callback);    // botão pressionado
elemento.addEventListener('mouseup', callback);      // botão solto
elemento.addEventListener('mousemove', callback);    // mouse se movendo
elemento.addEventListener('mouseenter', callback);   // mouse entra
elemento.addEventListener('mouseleave', callback);   // mouse sai
elemento.addEventListener('mouseover', callback);    // mouse sobre (propaga)
elemento.addEventListener('mouseout', callback);     // mouse sai (propaga)

// Teclado
elemento.addEventListener('keydown', callback);      // tecla pressionada
elemento.addEventListener('keyup', callback);        // tecla solta
elemento.addEventListener('keypress', callback);     // tecla caractere (depreca

// Formulário
form.addEventListener('submit', callback);           // envio do form
input.addEventListener('input', callback);           // valor mudando
input.addEventListener('change', callback);          // valor mudou (blur)
input.addEventListener('focus', callback);           // recebeu foco
input.addEventListener('blur', callback);            // perdeu foco

// Janela
window.addEventListener('load', callback);           // página carregou completa
window.addEventListener('DOMContentLoaded', callback); // DOM carregou (mais rá
window.addEventListener('resize', callback);         // janela redimensionada
window.addEventListener('scroll', callback);         // página rolou

// Outros
elemento.addEventListener('contextmenu', callback); // clique direito
elemento.addEventListener('wheel', callback);        // scroll do mouse
```

Objeto Event

javascript

```
botao.addEventListener('click', function(event) {
    // event ou e: objeto com informações do evento

    console.log(event.type);         // tipo do evento: "click"
    console.log(event.target);       // elemento que disparou
    console.log(event.currentTarget); // elemento com o listener

    // Prevenir comportamento padrão
    event.preventDefault(); // ex: evitar submit do form

    // Parar propagação (bubbling)
    event.stopPropagation();
});

// Eventos de mouse
elemento.addEventListener('click', function(e) {
    console.log(e.clientX, e.clientY); // posição do mouse
    console.log(e.button);             // qual botão (0=esquerdo, 1=meio, 2=dire
    console.log(e.ctrlKey);            // Ctrl pressionado?
    console.log(e.shiftKey);           // Shift pressionado?
    console.log(e.altKey);             // Alt pressionado?
});

// Eventos de teclado
input.addEventListener('keydown', function(e) {
    console.log(e.key);      // tecla: "a", "Enter", "Escape"
    console.log(e.code);     // código físico: "KeyA", "Enter"
    console.log(e.keyCode);  // código numérico (deprecated)

    // Verificar teclas específicas
    if (e.key === 'Enter') {
        console.log('Enter pressionado!');
    }

    if (e.key === 'Escape') {
        fecharModal();
    }
});

// Eventos de formulário
form.addEventListener('submit', function(e) {
    e.preventDefault(); // IMPORTANTE: evita recarregar página

    // Processar dados do formulário
```

```
    let formData = new FormData(form);
    console.log(formData.get('nome'));
});
```

Event Delegation (Delegação)

javascript

```
// Em vez de adicionar listener em cada item
// Adiciona no pai e verifica target

let lista = document.getElementById('lista');

lista.addEventListener('click', function(e) {
    // Verifica se clicou em um <li>
    if (e.target.tagName === 'LI') {
        console.log('Item clicado:', e.target.textContent);
        e.target.classList.toggle('selecionado');
    }

    // Ou usando closest
    let item = e.target.closest('.item');
    if (item) {
        console.log('Item encontrado');
    }
});

// Vantagem: funciona com elementos adicionados dinamicamente!
```

12. Timers e Assincronismo

setTimeout e setInterval

javascript

```javascript
// setTimeout: executa uma vez após delay
setTimeout(function() {
    console.log('Executou após 2 segundos');
}, 2000); // tempo em milissegundos

// Com arrow function
setTimeout(() => {
    console.log('Olá!');
}, 1000);

// Passar parâmetros
function saudar(nome, idade) {
    console.log(`Olá ${nome}, você tem ${idade} anos`);
}
setTimeout(saudar, 2000, 'João', 30);

// Cancelar timeout
let timer = setTimeout(() => {
    console.log('Isso não vai executar');
}, 3000);
clearTimeout(timer);

// setInterval: executa repetidamente
let contador = 0;
let intervalo = setInterval(function() {
    contador++;
    console.log(contador);

    if (contador === 5) {
        clearInterval(intervalo); // para o intervalo
    }
}, 1000); // a cada 1 segundo

// Exemplo: relógio
function atualizarRelogio() {
    let agora = new Date();
    let horas = agora.getHours().toString().padStart(2, '0');
    let minutos = agora.getMinutes().toString().padStart(2, '0');
    let segundos = agora.getSeconds().toString().padStart(2, '0');

    document.getElementById('relogio').textContent =
        `${horas}:${minutos}:${segundos}`;
}
```

```javascript
setInterval(atualizarRelogio, 1000);
```

13. JSON

```javascript
// JSON: JavaScript Object Notation
// Formato de texto para troca de dados

// Objeto JavaScript
let pessoa = {
    nome: 'João',
    idade: 30,
    ativo: true,
    hobbies: ['leitura', 'música']
};

// Converter objeto para JSON (string)
let json = JSON.stringify(pessoa);
console.log(json);
// '{"nome":"João","idade":30,"ativo":true,"hobbies":["leitura","música"]}'

// Converter JSON para objeto
let objetoNovamente = JSON.parse(json);
console.log(objetoNovamente.nome); // "João"

// Formatação bonita
let jsonFormatado = JSON.stringify(pessoa, null, 2); // 2 = espaços de indentaçã
console.log(jsonFormatado);

// Exemplo com localStorage
localStorage.setItem('usuario', JSON.stringify(pessoa));
let usuarioRecuperado = JSON.parse(localStorage.getItem('usuario'));
```

14. LocalStorage e SessionStorage

```javascript
```

```javascript
// localStorage: persiste mesmo após fechar navegador
// sessionStorage: persiste apenas durante a sessão

// Salvar
localStorage.setItem('nome', 'João');
localStorage.setItem('idade', '30');

// Ler
let nome = localStorage.getItem('nome');
console.log(nome); // "João"

// Remover
localStorage.removeItem('nome');

// Limpar tudo
localStorage.clear();

// Verificar se existe
if (localStorage.getItem('nome')) {
    console.log('Nome existe');
}

// Salvar objetos (precisa stringify)
let usuario = { nome: 'Maria', idade: 25 };
localStorage.setItem('usuario', JSON.stringify(usuario));

// Recuperar objetos
let usuarioRecuperado = JSON.parse(localStorage.getItem('usuario'));
console.log(usuarioRecuperado.nome); // "Maria"

// sessionStorage: mesmas funções
sessionStorage.setItem('temporario', 'valor');

// Tamanho (varia por navegador, geralmente ~5-10MB)
```

15. Tratamento de Erros

javascript

```javascript
// Try-Catch: captura erros
try {
    // Código que pode dar erro
    let resultado = funcaoQueNaoExiste();
    console.log(resultado);
} catch (erro) {
    // Executa se houver erro
    console.log('Erro capturado:', erro.message);
} finally {
    // Sempre executa (opcional)
    console.log('Finalizou');
}

// Lançar erro manualmente
function dividir(a, b) {
    if (b === 0) {
        throw new Error('Divisão por zero não permitida');
    }
    return a / b;
}

try {
    let resultado = dividir(10, 0);
} catch (e) {
    console.log(e.message); // "Divisão por zero não permitida"
}

// Tipos de erro
throw new Error('Erro genérico');
throw new TypeError('Tipo incorreto');
throw new RangeError('Valor fora do intervalo');
throw new ReferenceError('Referência inválida');
```

16. Operações Matemáticas

javascript

```javascript
// Objeto Math
Math.PI;            // 3.141592653589793
Math.E;             // 2.718281828459045

// Arredondamento
Math.round(4.7);    // 5 (arredonda)
Math.round(4.4);    // 4
Math.ceil(4.1);     // 5 (arredonda pra cima)
Math.floor(4.9);    // 4 (arredonda pra baixo)
Math.trunc(4.9);    // 4 (remove decimais)

// Valor absoluto
Math.abs(-5);       // 5

// Potência e raiz
Math.pow(2, 3);     // 8 (2³)
Math.sqrt(16);      // 4 (raiz quadrada)
Math.cbrt(27);      // 3 (raiz cúbica)

// Máximo e mínimo
Math.max(1, 5, 3, 9, 2);  // 9
Math.min(1, 5, 3, 9, 2);  // 1

// Aleatório
Math.random();      // número entre 0 e 1 (não inclui 1)

// Número aleatório entre min e max
function aleatorio(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}
console.log(aleatorio(1, 10)); // número de 1 a 10

// Trigonometria (radianos)
Math.sin(Math.PI / 2);  // 1
Math.cos(0);            // 1
Math.tan(Math.PI / 4);  // 1
```

17. Datas

javascript

```javascript
// Criar data
let agora = new Date();                    // data/hora atual
let especifica = new Date(2024, 0, 15);    // 15 jan 2024 (mês começa em 0!)
let comHora = new Date(2024, 0, 15, 14, 30, 0); // 15 jan 2024 14:30:00
let deString = new Date('2024-01-15');     // de string ISO
let deTimestamp = new Date(1705334400000); // de timestamp

// Obter componentes
let data = new Date();
data.getFullYear();     // 2024
data.getMonth();        // 0-11 (janeiro = 0!)
data.getDate();         // 1-31 (dia do mês)
data.getDay();          // 0-6 (domingo = 0)
data.getHours();        // 0-23
data.getMinutes();      // 0-59
data.getSeconds();      // 0-59
data.getMilliseconds(); // 0-999
data.getTime();         // timestamp (ms desde 1/1/1970)

// Definir componentes
data.setFullYear(2025);
data.setMonth(5);       // junho
data.setDate(15);
data.setHours(10);

// Formatação
data.toDateString();      // "Mon Jun 15 2025"
data.toTimeString();      // "10:00:00 GMT-0300"
data.toLocaleDateString(); // formato local
data.toLocaleTimeString(); // hora local
data.toISOString();       // "2025-06-15T13:00:00.000Z"

// Diferença entre datas
let inicio = new Date(2024, 0, 1);
let fim = new Date(2024, 11, 31);
let diferenca = fim - inicio; // em milissegundos
let dias = diferenca / (1000 * 60 * 60 * 24);
console.log(dias); // dias entre as datas
```

18. Expressões Regulares (RegEx) - Básico

javascript

```javascript
// Criar regex
let regex1 = /padrão/;
let regex2 = new RegExp('padrão');

// Flags comuns
let caseInsensitive = /texto/i;  // i = ignora maiúsculas/minúsculas
let global = /texto/g;           // g = encontra todas ocorrências
let multiline = /^texto/m;       // m = multilinha

// Métodos de teste
let texto = 'JavaScript é incrível';

// test: retorna true/false
let temJava = /Java/.test(texto);
console.log(temJava); // true

// exec: retorna array com match ou null
let match = /Script/.exec(texto);
console.log(match[0]); // "Script"

// Métodos de string
texto.match(/a/g);           // ["a", "a"] - todas ocorrências de 'a'
texto.search(/Script/);      // 4 - índice onde encontrou
texto.replace(/incrível/, 'fantástico'); // substitui
texto.split(/\s/);           // divide por espaços

// Padrões comuns
/\d/;            // dígito [0-9]
/\w/;            // caractere alfanumérico [a-zA-Z0-9_]
/\s/;            // espaço em branco
/./;             // qualquer caractere
/[abc]/;         // a, b ou c
/[0-9]/;         // qualquer dígito
/[a-z]/;         // qualquer letra minúscula
/^início/;       // começa com
/fim$/;          // termina com

// Quantificadores
/a+/;            // um ou mais 'a'
/a*/;            // zero ou mais 'a'
/a?/;            // zero ou um 'a'
/a{3}/;          // exatamente 3 'a'
/a{2,5}/;        // de 2 a 5 'a'
/a{2,}/;         // 2 ou mais 'a'
```

```javascript
// Exemplos práticos
let emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
let validEmail = emailRegex.test('user@example.com'); // true

let telefoneRegex = /^\(\d{2}\)\s?\d{4,5}-?\d{4}$/;
let validTelefone = telefoneRegex.test('(11) 98765-4321'); // true
```

19. Console (Debug)

javascript

```
// Mensagens
console.log('Mensagem normal');
console.info('Informação');
console.warn('Aviso');
console.error('Erro');

// Múltiplos argumentos
console.log('Valor:', 42, 'Texto:', 'olá');

// Formatação
let nome = 'João';
let idade = 30;
console.log(`Nome: ${nome}, Idade: ${idade}`);

// Objetos (mostra estrutura expandível)
let objeto = { nome: 'Maria', idade: 25 };
console.log(objeto);

// Table (exibe array/objeto como tabela)
let usuarios = [
    { nome: 'João', idade: 30 },
    { nome: 'Maria', idade: 25 }
];
console.table(usuarios);

// Tempo de execução
console.time('teste');
// código para medir
for (let i = 0; i < 1000000; i++) {}
console.timeEnd('teste'); // "teste: 5.2ms"

// Agrupar mensagens
console.group('Grupo 1');
console.log('Mensagem 1');
console.log('Mensagem 2');
console.groupEnd();

// Limpar console
console.clear();

// Assert (mostra erro se condição é falsa)
```

```
let x = 5;
console.assert(x > 10, 'x deveria ser maior que 10');
```

Integração das Três Tecnologias {#integração}

Exemplo Completo: Formulário Interativo

HTML:


html

```html
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <title>Cadastro de Usuário</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h1>Cadastro de Usuário</h1>

        <form id="formulario">
            <div class="campo">
                <label for="nome">Nome:</label>
                <input type="text" id="nome" required>
                <span class="erro" id="erroNome"></span>
            </div>

            <div class="campo">
                <label for="email">Email:</label>
                <input type="email" id="email" required>
                <span class="erro" id="erroEmail"></span>
            </div>

            <div class="campo">
                <label for="idade">Idade:</label>
                <input type="number" id="idade" min="0" max="120" required>
            </div>

            <button type="submit">Cadastrar</button>
        </form>

        <div id="resultado"></div>

        <h2>Usuários Cadastrados</h2>
        <ul id="listaUsuarios"></ul>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

CSS:

CSS

```css
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: Arial, sans-serif;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    min-height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
    padding: 20px;
}

.container {
    background-color: white;
    padding: 30px;
    border-radius: 10px;
    box-shadow: 0 10px 25px rgba(0, 0, 0, 0.2);
    max-width: 500px;
    width: 100%;
}

h1 {
    color: #333;
    margin-bottom: 20px;
    text-align: center;
}

h2 {
    color: #555;
    margin-top: 30px;
    margin-bottom: 15px;
    font-size: 1.3em;
}

.campo {
    margin-bottom: 20px;
}

label {
    display: block;
```

```
    margin-bottom: 5px;
    color: #555;
    font-weight: bold;
}

input {
    width: 100%;
    padding: 10px;
    border: 2px solid #ddd;
    border-radius: 5px;
    font-size: 16px;
    transition: border-color 0.3s;
}

input:focus {
    outline: none;
    border-color: #667eea;
}

input.invalido {
    border-color: #e74c3c;
}

.erro {
    color: #e74c3c;
    font-size: 14px;
    display: block;
    margin-top: 5px;
}

button {
    width: 100%;
    padding: 12px;
    background-color: #667eea;
    color: white;
    border: none;
    border-radius: 5px;
    font-size: 16px;
    font-weight: bold;
    cursor: pointer;
    transition: background-color 0.3s, transform 0.1s;
}

button:hover {
```

```
    background-color: #5568d3;
}

button:active {
    transform: scale(0.98);
}

#resultado {
    margin-top: 20px;
    padding: 15px;
    border-radius: 5px;
    display: none;
}

#resultado.sucesso {
    background-color: #d4edda;
    color: #155724;
    border: 1px solid #c3e6cb;
    display: block;
}

#listaUsuarios {
    list-style: none;
}

#listaUsuarios li {
    background-color: #f8f9fa;
    padding: 12px;
    margin-bottom: 10px;
    border-radius: 5px;
    border-left: 4px solid #667eea;
    display: flex;
    justify-content: space-between;
    align-items: center;
}

#listaUsuarios li button {
    width: auto;
    padding: 5px 10px;
    background-color: #e74c3c;
    font-size: 14px;
}

#listaUsuarios li button:hover {
```

```
    background-color: #c0392b;
}

@media (max-width: 600px) {
    .container {
        padding: 20px;
    }

    h1 {
        font-size: 1.5em;
    }
}
```

JavaScript:

```
javascript
```

```javascript
// Selecionar elementos
const formulario = document.getElementById('formulario');
const inputNome = document.getElementById('nome');
const inputEmail = document.getElementById('email');
const inputIdade = document.getElementById('idade');
const resultado = document.getElementById('resultado');
const listaUsuarios = document.getElementById('listaUsuarios');

// Array para armazenar usuários
let usuarios = [];

// Carregar usuários do localStorage
function carregarUsuarios() {
    const usuariosLocal = localStorage.getItem('usuarios');
    if (usuariosLocal) {
        usuarios = JSON.parse(usuariosLocal);
        renderizarLista();
    }
}

// Salvar usuários no localStorage
function salvarUsuarios() {
    localStorage.setItem('usuarios', JSON.stringify(usuarios));
}

// Validar email
function validarEmail(email) {
    const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return regex.test(email);
}

// Mostrar mensagem de resultado
function mostrarResultado(mensagem, tipo) {
    resultado.textContent = mensagem;
    resultado.className = tipo;
    resultado.style.display = 'block';

    setTimeout(() => {
        resultado.style.display = 'none';
    }, 3000);
}

// Renderizar lista de usuários
function renderizarLista() {
```

```
    listaUsuarios.innerHTML = '';

    usuarios.forEach((usuario, indice) => {
        const li = document.createElement('li');

        const info = document.createElement('div');
        info.innerHTML = `
            <strong>${usuario.nome}</strong><br>
            ${usuario.email} - ${usuario.idade} anos
        `;

        const btnRemover = document.createElement('button');
        btnRemover.textContent = 'Remover';
        btnRemover.addEventListener('click', () => removerUsuario(indice));

        li.appendChild(info);
        li.appendChild(btnRemover);
        listaUsuarios.appendChild(li);
    });
}

// Remover usuário
function removerUsuario(indice) {
    usuarios.splice(indice, 1);
    salvarUsuarios();
    renderizarLista();
    mostrarResultado('Usuário removido com sucesso!', 'sucesso');
}

// Event listener do formulário
formulario.addEventListener('submit', function(e) {
    e.preventDefault();

    // Pegar valores
    const nome = inputNome.value.trim();
    const email = inputEmail.value.trim();
    const idade = parseInt(inputIdade.value);

    // Validações
    let valido = true;

    if (nome.length < 3) {
        document.getElementById('erroNome').textContent =
            'Nome deve ter pelo menos 3 caracteres';
```

```
        inputNome.classList.add('invalido');
        valido = false;
    } else {
        document.getElementById('erroNome').textContent = '';
        inputNome.classList.remove('invalido');
    }

    if (!validarEmail(email)) {
        document.getElementById('erroEmail').textContent =
            'Email inválido';
        inputEmail.classList.add('invalido');
        valido = false;
    } else {
        document.getElementById('erroEmail').textContent = '';
        inputEmail.classList.remove('invalido');
    }

    if (!valido) {
        return;
    }

    // Criar usuário
    const usuario = {
        id: Date.now(),
        nome: nome,
        email: email,
        idade: idade
    };

    // Adicionar ao array
    usuarios.push(usuario);

    // Salvar e renderizar
    salvarUsuarios();
    renderizarLista();

    // Limpar formulário
    formulario.reset();

    // Mostrar mensagem
    mostrarResultado('Usuário cadastrado com sucesso!', 'sucesso');
});

// Validação em tempo real
```

```
inputNome.addEventListener('input', function() {
    if (inputNome.value.trim().length >= 3) {
        inputNome.classList.remove('invalido');
        document.getElementById('erroNome').textContent = '';
    }
});

inputEmail.addEventListener('input', function() {
    if (validarEmail(inputEmail.value.trim())) {
        inputEmail.classList.remove('invalido');
        document.getElementById('erroEmail').textContent = '';
    }
});

// Carregar usuários ao iniciar
carregarUsuarios();
```

Boas Práticas e Dicas para Prova {#dicas}

HTML - Pontos Importantes

1. **Sempre use `<!DOCTYPE html>`** no início
2. **Atributo `alt` em imagens** é obrigatório
3. **Use tags semânticas** (header, nav, main, article, section, aside, footer)
4. **Formulários:** sempre use `<label>` com `for` vinculado ao `id` do input
5. **`name` vs `id`:** `name` para envio de formulário, `id` para JavaScript/CSS
6. **Apenas um `<h1>` por página** (boas práticas SEO)
7. **Links externos:** considere usar `target="_blank"` e `rel="noopener"`
8. **Caracteres especiais:** use entidades HTML (<, >, &, etc)
9. **Validação HTML:** use atributos como `required`, `min`, `max`, `pattern`
10. **Meta viewport** é essencial para responsividade

CSS - Pontos Importantes

1. **Especificidade:** ID > Classe > Elemento (evite `!important`)
2. **Box-sizing:** use `box-sizing: border-box` em tudo
3. **Seletores:** prefira classes a IDs para estilização
4. **CSS externo** é melhor que inline ou interno
5. **Mobile First:** comece pelos estilos mobile, depois adicione media queries
6. **Flexbox:** ferramenta principal para layouts modernos
7. **Position absolute** precisa de um pai com `position: relative`
8. **Z-index** só funciona com elementos posicionados (não static)
9. **Transitions** precisam de propriedade inicial definida
10. **Normalize/Reset CSS:** considere usar para consistência entre navegadores

JavaScript - Pontos Importantes

1. **Use `const` por padrão**, `let` quando precisar reatribuir, evite `var`
2. **Sempre use `===` e `!==`** (igualdade estrita)
3. **`addEventListener`** é melhor que atributos HTML (onclick, etc)

4. **`e.preventDefault()`** em forms evita recarregar a página
5. **Array methods** (map, filter, reduce) são muito cobrados
6. **`this`** muda de acordo com o contexto
7. **Arrow functions** não têm `this` próprio
8. **Template strings** (backticks) permitem interpolação
9. **Desestruturação** simplifica extração de dados
10. **`querySelector`** é mais flexível que `getElementById`

Erros Comuns a Evitar

HTML

html

```
<!-- ERRADO -->
<img src="foto.jpg"> <!-- falta alt -->
<input type="text"> <label>Nome</label> <!-- label desconectado -->
<div><p>Texto</div></p> <!-- tags cruzadas -->

<!-- CORRETO -->
<img src="foto.jpg" alt="Descrição">
<label for="nome">Nome</label>
<input type="text" id="nome">
<div><p>Texto</p></div>
```

CSS

css

```css
/* ERRADO */
.classe {
    width: 200px;
    padding: 20px; /* total = 240px se não usar box-sizing */
}

#id {
    color: blue !important; /* evite !important */
}

/* CORRETO */
* {
    box-sizing: border-box;
}

.classe {
    width: 200px;
    padding: 20px; /* total = 200px */
}

.classe {
    color: blue; /* sem !important */
}
```

JavaScript

```
javascript
```

```javascript
// ERRADO
if (x = 5) { } // atribuição em vez de comparação
if (x == "5") { } // comparação frouxa

var nome = "João"; // var tem problemas de escopo

elemento.onclick = function() {}; // sobrescreve listeners anteriores

// CORRETO
if (x === 5) { } // comparação estrita
if (x === "5") { } // comparação estrita

const nome = "João"; // const/let têm escopo de bloco

elemento.addEventListener('click', function() {}); // múltiplos listeners possí
```

Padrões de Código Comuns

Validação de Formulário

javascript

```javascript
function validarFormulario() {
    const nome = document.getElementById('nome').value.trim();
    const email = document.getElementById('email').value.trim();

    let erros = [];

    if (nome.length < 3) {
        erros.push('Nome deve ter pelo menos 3 caracteres');
    }

    const regexEmail = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!regexEmail.test(email)) {
        erros.push('Email inválido');
    }

    if (erros.length > 0) {
        alert(erros.join('\n'));
        return false;
    }

    return true;
}

formulario.addEventListener('submit', function(e) {
    e.preventDefault();
    if (validarFormulario()) {
        // Processar dados
    }
});
```

Toggle de Classe (Mostrar/Ocultar)

javascript

```javascript
const botao = document.getElementById('botao');
const menu = document.getElementById('menu');

botao.addEventListener('click', function() {
    menu.classList.toggle('ativo');
});
```

Criar Elementos Dinamicamente

javascript

```javascript
function adicionarItem(texto) {
    const li = document.createElement('li');
    li.textContent = texto;
    li.addEventListener('click', function() {
        this.remove();
    });

    document.getElementById('lista').appendChild(li);
}
```

Fazer Contagem Regressiva

javascript

```javascript
let tempo = 60; // 60 segundos
const display = document.getElementById('timer');

const intervalo = setInterval(function() {
    tempo--;
    display.textContent = tempo;

    if (tempo <= 0) {
        clearInterval(intervalo);
        alert('Tempo esgotado!');
    }
}, 1000);
```

Filtrar Lista

javascript

```javascript
const input = document.getElementById('busca');
const itens = document.querySelectorAll('.item');

input.addEventListener('input', function() {
    const termo = input.value.toLowerCase();

    itens.forEach(item => {
        const texto = item.textContent.toLowerCase();
        if (texto.includes(termo)) {
            item.style.display = 'block';
        } else {
            item.style.display = 'none';
        }
    });
});
```

Atalhos e Truques Úteis

JavaScript

javascript

```javascript
// Converter string para número
let num = +"42"; // 42
let num2 = Number("42"); // 42
let num3 = parseInt("42"); // 42

// Verificar se é número
isNaN("texto"); // true
isNaN(42); // false

// Valores padrão
let nome = usuario.nome || "Anônimo";
let idade = usuario.idade ?? 18; // só usa padrão se null/undefined

// Copiar array
let copia = [...original];
let copia2 = original.slice();

// Copiar objeto
let copia = { ...original };
let copia2 = Object.assign({}, original);

// Remover duplicatas de array
let unicos = [...new Set([1, 2, 2, 3, 3, 4])]; // [1, 2, 3, 4]

// Inverter string
let invertida = "hello".split('').reverse().join(''); // "olleh"

// Gerar array de números
let numeros = Array.from({length: 5}, (_, i) => i + 1); // [1,2,3,4,5]
let numeros2 = [...Array(5)].map((_, i) => i + 1); // [1,2,3,4,5]

// Somar array
let soma = [1, 2, 3, 4, 5].reduce((acc, val) => acc + val, 0); // 15

// Encontrar máximo em array
let maximo = Math.max(...[1, 5, 3, 9, 2]); // 9

// Embaralhar array
function embaralhar(array) {
    return array.sort(() => Math.random() - 0.5);
}

// Debounce (atrasa execução)
function debounce(func, delay) {
```

```
    let timeout;
    return function(...args) {
        clearTimeout(timeout);
        timeout = setTimeout(() => func.apply(this, args), delay);
    };
}

// Uso do debounce
input.addEventListener('input', debounce(function() {
    console.log('Pesquisando...', this.value);
}, 500));
```

CSS

CSS

```css
/* Centralizar absolutamente */
.centralizado {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
}

/* Centralizar com Flexbox */
.flex-centro {
    display: flex;
    justify-content: center;
    align-items: center;
}

/* Truncar texto longo */
.truncar {
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
}

/* Proporção de aspecto (quadrado) */
.quadrado {
    width: 100%;
    padding-bottom: 100%; /* altura = largura */
    position: relative;
}

/* Esconder visualmente mas manter acessível */
.sr-only {
    position: absolute;
    width: 1px;
    height: 1px;
    padding: 0;
    margin: -1px;
    overflow: hidden;
    clip: rect(0, 0, 0, 0);
    border: 0;
}

/* Remover aparência de botão/input */
.sem-estilo {
    background: none;
```

```
    border: none;
    padding: 0;
    margin: 0;
    font: inherit;
    cursor: pointer;
}

/* Smooth scroll */
html {
    scroll-behavior: smooth;
}

/* Estilizar scrollbar (webkit) */
::-webkit-scrollbar {
    width: 10px;
}

::-webkit-scrollbar-track {
    background: #f1f1f1;
}

::-webkit-scrollbar-thumb {
    background: #888;
    border-radius: 5px;
}
```

Checklist para Prova

Antes de Entregar o Código

HTML:

- [ ] Tem `<!DOCTYPE html>`?
- [ ] Tag `<html>` tem `lang="pt-BR"`?
- [ ] Tem `<meta charset="UTF-8">`?
- [ ] Tem `<meta name="viewport">`?
- [ ] Todas imagens têm `alt`?
- [ ] Formulários têm labels associados?
- [ ] IDs são únicos?
- [ ] Tags estão fechadas corretamente?

CSS:

- [ ] Usei classes em vez de IDs para estilização?
- [ ] Está em arquivo externo (não inline)?
- [ ] Usei `box-sizing: border-box`?
- [ ] Media queries estão corretas?

- [ ] Cores têm contraste adequado?
- [ ] Não abusei de `!important`?

JavaScript:

- [ ] Usei `const`/`let` em vez de `var`?
- [ ] Usei `===` em vez de `==`?
- [ ] Event listeners estão corretos?
- [ ] Usei `preventDefault()` em forms?
- [ ] Tratei possíveis erros?
- [ ] Variáveis têm nomes descritivos?
- [ ] Código está comentado onde necessário?
- [ ] Não tem `console.log()` desnecessários?

Perguntas Teóricas Comuns

1. Qual a diferença entre HTML, CSS e JavaScript?

- HTML: estrutura e conteúdo
- CSS: apresentação e estilo visual
- JavaScript: comportamento e interatividade

2. O que é o DOM? Document Object Model - representação em árvore do documento HTML que pode ser manipulada com JavaScript.

3. Diferença entre `display: none` e `visibility: hidden`?

- `display: none`: remove completamente, não ocupa espaço
- `visibility: hidden`: esconde mas mantém espaço

4. O que é event bubbling? Eventos propagam do elemento filho para os pais. Pode ser parado com `stopPropagation()`.

5. Diferença entre `let`, `const` e `var`?

- `let`: escopo de bloco, pode reatribuir
- `const`: escopo de bloco, não pode reatribuir
- `var`: escopo de função, sofre hoisting (evitar)

6. O que são arrow functions? Sintaxe curta de funções ES6: `(params) => expressão`. Não têm `this` próprio.

7. Diferença entre == e ===?

- `==`: compara valores, faz conversão de tipo
- `===`: compara valores E tipos (use sempre!)

8. O que é localStorage? Armazenamento no navegador que persiste mesmo após fechar. Máximo ~5-10MB.

9. O que é JSON? JavaScript Object Notation - formato de texto para trocar dados. Use `JSON.stringify()` e `JSON.parse()`.

10. Diferença entre `position: relative` e `absolute`?

- `relative`: relativo à posição original
- `absolute`: relativo ao ancestral posicionado

Exercícios Rápidos para Treinar

Exercício 1: Criar Lista de Tarefas

html

```html
<!-- HTML -->
<input type="text" id="tarefa" placeholder="Nova tarefa">
<button id="adicionar">Adicionar</button>
<ul id="lista"></ul>
```

javascript

```javascript
// JavaScript
const input = document.getElementById('tarefa');
const botao = document.getElementById('adicionar');
const lista = document.getElementById('lista');

function adicionarTarefa() {
    const texto = input.value.trim();
    if (texto === '') return;

    const li = document.createElement('li');
    li.textContent = texto;

    li.addEventListener('click', function() {
        this.remove();
    });

    lista.appendChild(li);
    input.value = '';
}

botao.addEventListener('click', adicionarTarefa);
input.addEventListener('keypress', function(e) {
    if (e.key === 'Enter') {
        adicionarTarefa();
    }
});
```

Exercício 2: Contador

```html
<div id="contador">0</div>
<button id="incrementar">+</button>
<button id="decrementar">-</button>
<button id="resetar">Reset</button>
```

```javascript
let count = 0;
const display = document.getElementById('contador');

document.getElementById('incrementar').addEventListener('click', () => {
    count++;
    display.textContent = count;
});

document.getElementById('decrementar').addEventListener('click', () => {
    count--;
    display.textContent = count;
});

document.getElementById('resetar').addEventListener('click', () => {
    count = 0;
    display.textContent = count;
});
```

Exercício 3: Calculadora Simples

```html
```

```
<input type="number" id="num1">
<select id="operacao">
    <option value="+">+</option>
    <option value="-">-</option>
    <option value="*">×</option>
    <option value="/">÷</option>
</select>
<input type="number" id="num2">
<button id="calcular">=</button>
<div id="resultado"></div>
```

javascript

```
document.getElementById('calcular').addEventListener('click', function() {
    const num1 = parseFloat(document.getElementById('num1').value);
    const num2 = parseFloat(document.getElementById('num2').value);
    const operacao = document.getElementById('operacao').value;

    let resultado;

    switch(operacao) {
        case '+':
            resultado = num1 + num2;
            break;
        case '-':
            resultado = num1 - num2;
            break;
        case '*':
            resultado = num1 * num2;
            break;
        case '/':
            resultado = num2 !== 0 ? num1 / num2 : 'Erro: divisão por zero';
            break;
    }

    document.getElementById('resultado').textContent = resultado;
});
```

Exercício 4: Mudar Cor de Fundo

html

```
<button id="mudarCor">Mudar Cor</button>
```

javascript

```
document.getElementById('mudarCor').addEventListener('click', function() {
    const cores = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#FFA07A', '#98D8C8', '#F7D
    const corAleatoria = cores[Math.floor(Math.random() * cores.length)];
    document.body.style.backgroundColor = corAleatoria;
});
```

Exercício 5: Validação de Senha

html

```
<input type="password" id="senha">
<div id="forca"></div>
```

javascript

```
const senha = document.getElementById('senha');
const forca = document.getElementById('forca');

senha.addEventListener('input', function() {
    const valor = senha.value;
    let nivel = 0;

    if (valor.length >= 8) nivel++;
    if (/[a-z]/.test(valor)) nivel++;
    if (/[A-Z]/.test(valor)) nivel++;
    if (/[0-9]/.test(valor)) nivel++;
    if (/[^a-zA-Z0-9]/.test(valor)) nivel++;

    if (nivel <= 2) {
        forca.textContent = 'Fraca';
        forca.style.color = 'red';
    } else if (nivel <= 3) {
        forca.textContent = 'Média';
        forca.style.color = 'orange';
    } else {
        forca.textContent = 'Forte';
        forca.style.color = 'green';
    }
});
```

Dicas Finais

1. **Leia o enunciado com atenção** - entenda exatamente o que é pedido
2. **Comece pelo HTML** - estruture antes de estilizar
3. **Teste constantemente** - não deixe para testar só no final
4. **Use nomes descritivos** - `btnSubmit` é melhor que `btn1`
5. **Comente código complexo** - ajuda você e o avaliador
6. **Não se preocupe com perfeição visual** - funcionalidade primeiro
7. **Console.log é seu amigo** - use para debug
8. **Mantenha o código organizado** - indentação correta
9. **Releia seu código** - pegue erros bobos antes de entregar
10. **Gerencie seu tempo** - não fique preso em um problema só

Recursos de Referência Rápida

Consulta rápida de métodos:

Arrays: push, pop, shift, unshift, slice, splice, forEach, map, filter, find, reduce, sort, reverse, indexOf, includes

Strings: length, toUpperCase, toLowerCase, trim, split, slice, substring, replace, includes, startsWith, endsWith, charAt

Objetos: Object.keys(), Object.values(), Object.entries(), Object.assign()

DOM: getElementById, querySelector, querySelectorAll, addEventListener, createElement, appendChild, remove, classList, innerHTML, textContent, value

Eventos comuns: click, submit, input, change, focus, blur, keydown, keyup, load, DOMContentLoaded

---

Conclusão

Este guia cobre todos os fundamentos essenciais de HTML, CSS e JavaScript para programação web. Pratique os exemplos, refaça os exercícios e teste variações dos códigos apresentados.

Lembre-se:

- HTML estrutura o conteúdo
- CSS estiliza a apresentação
- JavaScript adiciona interatividade
- As três tecnologias trabalham juntas para criar páginas web completas

Para a prova:

- Entenda os conceitos, não apenas decore
- Pratique escrevendo código à mão
- Teste diferentes combinações
- Leia mensagens de erro com atenção
- Mantenha a calma e pense logicamente

Boa sorte na sua prova! Você consegue! 🚀