

HTML5

1. Fundamentos do HTML

- **Estrutura básica de uma página HTML:** Uma página HTML começa com a declaração do tipo de documento (`<!DOCTYPE html>`) e contém a estrutura básica que inclui as seções `<head>` e `<body>`.

```
html
<!DOCTYPE html>
<html>
<head>
    <title>Minha Página</title>
</head>
<body>
    <h1>Bem-vindo à minha página!</h1>
</body>
</html>
```

- **Elementos e tags HTML:** HTML é composto de "tags" que delimitam elementos. Exemplo: `<p>` para parágrafos.

```
html
<p>Este é um parágrafo de exemplo.</p>
```

- **Atributos e valores:** As tags HTML podem ter atributos que fornecem informações adicionais sobre o elemento. Exemplo: ``.

```
html

```

- **Comentários em HTML:** Comentários são inseridos com `<!-- comentário aqui -->` e não aparecem na visualização do navegador.

```
html
<!-- Este é um comentário em HTML -->
```

- **UTF-8 e caracteres especiais:** O HTML deve estar codificado em UTF-8 para suportar caracteres especiais. Entidades como `&nbsp` e `&lt` representam espaços em branco e o sinal de menor, respectivamente.

```
html
&ampnbsp <!-- representa um espaço em branco -->
&amplt!-- representa o caractere menor (<) -->
```

2. Estruturas de Documentos

- **Elementos de título (`<head>`, `<title>`):** A seção `<head>` contém metadados, como o `<title>` que aparece na aba do navegador e pode incluir links para CSS e scripts.

```
html
<head>
    <title>Exemplo de Título</title>
    <link rel="stylesheet" href="styles.css">
</head>
```

- **Elemento do corpo (`<body>`):** Contém todo o conteúdo visível da página, como texto, imagens e outros elementos.

```
html
<body>
    <h1>Página Principal</h1>
    <p>Conteúdo visível aqui.</p>
</body>
```

- **Divisores e seções:** Elementos como `<div>`, `<section>`, `<article>`, `<aside>`, `<header>`, `<footer>`, e `<nav>` ajudam a estruturar a página semanticamente.

```
html
<header>
    <h1>Título do Site</h1>
    <nav>
        <ul>
            <li><a href="#home">Início</a></li>
            <li><a href="#sobre">Sobre</a></li>
        </ul>
    </nav>
</header>
```

3. Textos e Títulos

- **Títulos (`<h1>` a `<h6>`):** Usado para criar cabeçalhos, com `<h1>` como o mais importante e `<h6>` o menos importante.

```
html
<h1>Título Principal</h1>
<h2>Subtítulo</h2>
```

- **Parágrafos (`<p>`):** Criação de blocos de texto.

```
html
<p>Este é um parágrafo com texto explicativo.</p>
```

- **Quebra de linha e parágrafos:** O `
` insere uma quebra de linha, enquanto `<hr>` insere uma linha horizontal.

```
html
Linha 1.<br>Linha 2.
<hr>
```

- **Citações e referências:** `<blockquote>` para citações longas, `<cite>` para referenciar obras e `<q>` para citações curtas.

```
html
<blockquote>Este é um trecho de uma citação longa.</blockquote>
<p>De acordo com <cite>Autor, Título</cite>.</p>
```

4. Listas

- **Listas ordenadas (``):** Cria listas numeradas.

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

- **Listas não ordenadas (``):** Cria listas com marcadores.

```
html
<ul>
    <li>Item A</li>
    <li>Item B</li>
    <li>Item C</li>
</ul>
```

- **Listas de definição (<dl>, <dt>, <dd>):** Para termos e suas definições, com <dt> para o termo e <dd> para a definição.

```
html
<dl>
    <dt>HTML</dt>
    <dd>Linguagem de marcação para criar páginas web.</dd>
    <dt>CSS</dt>
    <dd>Estilos para a apresentação de páginas HTML.</dd>
</dl>
```

5. Links e Ancoragens

- **Hyperlinks (<a>):** Usado para criar links, onde o atributo href define o destino do link.

```
html
<a href="https://www.example.com">Visite nosso site!</a>
```

- **Ancoragens:** Um link pode direcionar para um ponto específico na mesma página usando name ou id.

```
html
<a href="#section1">Ir para Seção 1</a>

<h2 id="section1">Seção 1</h2>
<p>Conteúdo da Seção 1.</p>
```

- **Links externos e internos:** Links podem levar a outros sites ou para diferentes partes da mesma página.

```
html
<a href="https://www.google.com">Link externo</a>
<a href="#secao2">Link interno para outra parte da página</a>
```

6. Imagens e Mídia

- **Inserindo imagens ():** Usa o atributo src para o caminho da imagem e alt para a descrição.

```
html

```

- **Mídia em HTML5:** Elementos <audio> e <video> permitem a inclusão de mídia. track é usado dentro de <video> para legendas.

```
html
<audio controls>
    <source src="audio.mp3" type="audio/mpeg">
        Seu navegador não suporta o elemento de áudio.
</audio>

<video width="320" height="240" controls>
```

```
<source src="video.mp4" type="video/mp4">
<track src="legendas.vtt" kind="subtitles" srclang="pt" label="Português">
    Seu navegador não suporta o elemento de vídeo.
</video>
```

7. Formulários e Entrada de Dados

- **Estrutura de formulários (<form>):** Estruturas que permitem a coleta de dados do usuário.

```
html
<form action="/submit" method="post">
    <label for="nome">Nome:</label>
    <input type="text" id="nome" name="nome" required>
    <input type="submit" value="Enviar">
</form>
```

- **Tipos de entrada:** Usando <input>, <textarea>, <select> para criar diferentes tipos de campos para entrada de dados.

```
html
<input type="text" name="username" placeholder="Usuário">
<textarea name="mensagem" rows="4" cols="50"></textarea>
<select name="opcoes">
    <option value="opcao1">Opção 1</option>
    <option value="opcao2">Opção 2</option>
</select>
```

- **Atributos de validação:** Como required, pattern, e min/max para controlar a entrada.

```
html
<input type="email" name="email" required>
<input type="number" name="idade" min="1" max="120">
```

8. Semântica e Acessibilidade

- **Importância da semântica:** Usar elementos HTML de maneira semântica melhora a legibilidade e acessibilidade.

```
html
<article>
    <h2>Título do Artigo</h2>
    <p>Este é o conteúdo do artigo, que descreve um tópico relevante.</p>
</article>
```

- **ARIA:** Conjunto de atributos que melhoram a acessibilidade em aplicativos ricos. Exemplo do uso de ARIA:

```
html
<button aria-label="Fechar" onclick="fecharModal()">X</button>
```

- **Elementos semânticos:** Utilizar elementos apropriados para descrever a função, como <header>, <footer>, e <nav>.

```
html
<nav>
    <ul>
        <li><a href="#home">Home</a></li>
        <li><a href="#contato">Contato</a></li>
    </ul>
</nav>
```

9. Novos Elementos do HTML5

- **Elementos estruturais:** <main>, <figure>, e <figcaption> ajudam na organização do conteúdo.

```
html
<main>
  <article>
    <h1>Título do Artigo</h1>
    <figure>
      
      <figcaption>Esta é uma legenda para a imagem.</figcaption>
    </figure>
    <p>Conteúdo do artigo.</p>
  </article>
</main>
```

- **Elementos de capacidade (<progress>, <output>):** Usados para representar progresso de ações ou resultados de cálculos.

```
html
<progress value="50" max="100">50%</progress>
<output id="resultado">Resultado: </output>
```

10. APIs e Funcionalidades HTML5

- **APIs de armazenamento:** Permitem que dados sejam armazenados localmente no navegador (LocalStorage e Session Storage).

```
javascript
localStorage.setItem('chave', 'valor');
var valor = localStorage.getItem('chave');
```

- **Geolocalização:** API que permite obter a localização geográfica do usuário.

```
javascript
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(function(position) {
    console.log("Latitude: " + position.coords.latitude + ", Longitude: "
+ position.coords.longitude);
  });
}
```

- **Drag and Drop:** Funcionalidade que permite mover elementos da interface de forma interativa.

```
html
<div draggable="true" ondragstart="event.dataTransfer.setData('text/plain',
null)">Arraste-me!</div>
```

- **Canvas API:** Usada para desenhar gráficos, imagens e animações em tempo real dentro de uma página.

```
html
<canvas id="meuCanvas" width="200" height="100"></canvas>
<script>
  var canvas = document.getElementById('meuCanvas');
  var ctx = canvas.getContext('2d');
  ctx.fillStyle = "red";
  ctx.fillRect(10, 10, 150, 50);
</script>
```

- **Web Workers:** Permite ao JavaScript rodar em segundo plano, sem bloquear a interface do usuário.

```
javascript
var meuWorker = new Worker('worker.js');
meuWorker.postMessage('Começar');
```

11. Microdados e Estrutura de Dados

- **Microdados (data-* attributes):** Permitem adicionar metadata a elementos HTML, que pode ser lido por buscadores e outros serviços.

```
html
<div itemscope itemtype="http://schema.org/Produto">
    <span itemprop="nome">Produto Exemplo</span>
    <span itemprop="preco">R$ 100,00</span>
</div>
```

- **Schema.org e Rich Snippets:** Usado para melhorar a interpretação de dados pelos motores de busca.

```
html
<div itemscope itemtype="http://schema.org/Artigo">
    <h2 itemprop="headline">Título do Artigo</h2>
    <span itemprop="autor">Autor do Artigo</span>
    <time itemprop="dataPublicacao" datetime="2023-01-01">1 de Janeiro de 2023</time>
</div>
```

12. SEO e HTML5

- **Otimização para motores de busca:** Práticas que ajudam a melhorar o posicionamento da página nos resultados de busca. Isso inclui o uso correto de tags de título, meta descrições e cabeçalhos.

```
html
<head>
    <title>Como Aprender HTML</title>
    <meta name="description" content="Dicas e tutoriais sobre HTML para iniciantes.">
</head>
```

- **Atributos importantes para SEO:** Como `<title>`, `<meta>`, e `alt` nas imagens, que ajudam os buscadores a entender o conteúdo da página.

```
html

```

CSS3

1. Fundamentos do CSS

Como adicionar CSS a uma página HTML: O CSS pode ser adicionado de três maneiras principais: inline (dentro do elemento HTML usando o atributo `style`), interno (dentro de uma `<style>` na seção `<head>` do HTML), ou externo (linkando uma folha de estilo externa com `<link>`).

Exemplo:

```
html
<!-- Inline -->
<h1 style="color: blue;">Título em azul</h1>
```

```

<!-- Interno -->
<style>
    h1 { color: green; }
</style>

<!-- Externo -->
<link rel="stylesheet" href="styles.css">

```

Seletores CSS: Os seletores são usados para escolher quais elementos HTML receberão os estilos. Existem vários tipos, incluindo seletores de tag, classe, ID, descendente e mais.

Exemplo:

```

css

/* Seletores */
h1 { color: red; }           /* Seletor de tag */
.classe { font-size: 20px; }   /* Seletor de classe */
#id { margin: 10px; }         /* Seletor de ID */
div p { color: blue; }        /* Seletor descendente */

```

Cascata e ordem de aplicação de estilos: A cascata é a forma como as regras de estilo são aplicadas, considerando a origem (browser, usuário, autor) e a ordem das declarações no código.

Exemplo:

```

css

/* Este estilo será aplicado */
p { color: black; }

/* O estilo abaixo vai sobreescrivendo o anterior devido à ordem */
p { color: red; }

```

Especificidade e prioridade de regras: Especificidade é um conceito que determina quais regras CSS são aplicadas quando há múltiplas declarações para um mesmo elemento. Quão mais específica for uma regra, maior sua prioridade.

Exemplo:

```

css

.classe p { color: blue; } /* Especificidade = 0-1-1 */
#id p { color: green; }   /* Especificidade = 1-0-1 */
/* O texto em um <p> dentro de #id será verde */

```

Herança de propriedades: Algumas propriedades CSS são herdadas dos elementos pai para os elementos filhos. Exemplo: color e font-family são propriedades que normalmente são herdadas.

Exemplo:

```

css

div { color: purple; }
/* Um p dentro do div herdará a cor */
<div>
    <p>Texto aqui será roxo.</p>
</div>

```

Importação de folhas de estilo (@import): O @import permite que você importe uma folha de estilo dentro de outra. No entanto, deve ser usado com cuidado, pois pode impactar na performance do carregamento.

Exemplo:

```

css

@import url("styles2.css");

```

2. Trabalhando com Fundo (Background)

background-color: Define a cor de fundo de um elemento, podendo ser em formato hexadecimal, RGB, rgba, ou palavras-chave.

Exemplo:

```
css
body { background-color: #f0f0f0; } /* Cor hexadecimal */
```

background-image: Permite adicionar uma imagem como fundo. Pode ser um URL para um arquivo de imagem.

Exemplo:

```
css
div { background-image: url('imagem.jpg'); }
```

background-position: Determina a posição da imagem de fundo. Valores comuns incluem top, bottom, left, right, e valores em pixels ou porcentagens.

Exemplo:

```
css
div { background-image: url('imagem.jpg'); background-position: center; }
```

background-size: Controla o tamanho da imagem de fundo. Pode receber valores como cover (preencher todo o espaço) ou contain (manter a proporção sem estourar o contêiner).

Exemplo:

```
css
div { background-image: url('imagem.jpg'); background-size: cover; }
```

background-repeat: Define como a imagem de fundo é repetida. Os valores podem ser repeat, no-repeat, repeat-x, repeat-y.

Exemplo:

```
css
div { background-image: url('imagem.jpg'); background-repeat: no-repeat; }
```

background-attachment: Controla se a imagem de fundo se move com a rolagem da página (scroll) ou se fixa na tela (fixed).

Exemplo:

```
css
div { background-image: url('imagem.jpg'); background-attachment: fixed; }
```

background-blend-mode: Permite misturar a cor de fundo com a imagem de fundo usando modos de mesclagem, como multiply, screen, etc.

Exemplo:

```
css
div {
    background-color: rgba(255, 0, 0, 0.5);
    background-image: url('imagem.jpg');
    background-blend-mode: multiply;
}
```

background-clip: Determina até onde o fundo é renderizado dentro do elemento, podendo ser definido como border-box, padding-box ou content-box.

Exemplo:

```
css
div { background: red; background-clip: padding-box; }
```

background-origin: Define a área a partir da qual o fundo é calculado, podendo ser border-box, padding-box ou content-box.

Exemplo:

```
css
div { background-image: url('imagem.jpg'); background-origin: content-box; }
```

3. Bordas (Border)

border: Propriedade abreviada que configura a largura, estilo e cor da borda ao mesmo tempo.

Exemplo:

```
css
div { border: 2px solid black; }
```

border-style, border-width, border-color: Essas propriedades individuais permitem definir o estilo (sólido, tracejado, etc.), a largura, e a cor da borda.

Exemplo:

```
css
div {
    border-style: dashed;
    border-width: 3px;
    border-color: blue;
}
```

border-radius: Cria bordas arredondadas. Valor pode ser em pixels ou porcentagem, sendo que 50% resulta em círculos/perfis circulares.

Exemplo:

```
css
div { border-radius: 10px; } /* Bordas levemente arredondadas */
```

border-image: Permite usar uma imagem como borda, com opções para controlar o tamanho e o posicionamento da imagem.

Exemplo:

```
css
div { border-image: url('imagem.jpg') 30 stretch; }
```

4. Cores e Opacidade

color: Define a cor do texto. Assim como background-color, pode-se usar valores hexadecimais, RGB, RGBA, HSL etc.

Exemplo:

```
css
p { color: #ff5733; } /* Usando hexadecimal */
```

opacity: Controla a opacidade de um elemento. Um valor de 1 é completamente opaco, e 0 é completamente transparente.

Exemplo:

```
css
div { opacity: 0.5; } /* 50% de opacidade */
```

5. Texto e Tipografia

font-family: Especifica o tipo de fonte a ser usada. Pode-se definir uma fonte genérica para fallback.

Exemplo:

```
css
h1 { font-family: 'Arial', sans-serif; }
```

font-size: Ajusta o tamanho do texto, com unidades como px, em, rem.

Exemplo:

```
css
p { font-size: 16px; }
```

font-weight: Controla a espessura do texto, com valores como normal, bold, e valores numéricos (100-900).

Exemplo:

```
css
h2 { font-weight: bold; }
```

font-style: Define o estilo da fonte, permitindo que o texto fique itálico ou normal.

Exemplo:

```
css
em { font-style: italic; }
```

letter-spacing: Ajusta o espaço entre letras, podendo ser positivo ou negativo.

Exemplo:

```
css
h1 { letter-spacing: 1px; }
```

line-height: Define a altura da linha, que influencia o espaçamento vertical entre as linhas de texto.

Exemplo:

```
css
p { line-height: 1.5; }
```

text-align: Controla o alinhamento do texto (left, right, center, justify).

Exemplo:

```
css
p { text-align: center; }
```

text-decoration: Adiciona decoração ao texto, como sublinhado, tachado ou overline.

Exemplo:

```
css
a { text-decoration: underline; }
```

text-transform: Controla o caso do texto (como transformá-lo em maiúsculas ou minúsculas).

Exemplo:

```
css
h1 { text-transform: uppercase; }
```

6. Tamanhos e Dimensões

width, min-width, max-width: Controlam a largura de um elemento, definindo um valor padrão e limites inferior e superior.

Exemplo:

```
css
div { width: 100%; max-width: 600px; min-width: 300px; }
```

height, min-height, max-height: Funciona de forma semelhante à largura, mas para a altura do elemento.

Exemplo:

```
css
div { height: 200px; max-height: 400px; min-height: 100px; }
```

7. Espaçamentos

margin: Define o espaço exterior ao redor de um elemento. Os valores podem ser definidos individualmente para cada lado (topo, direito, fundo, esquerdo).

Exemplo:

```
css
div { margin: 20px; } /* Margens de 20px em todos os lados */
```

padding: Controla o espaçamento interno de um elemento, entre o conteúdo e a borda.

Exemplo:

```
css
div { padding: 10px; } /* 10px de padding em todos os lados */
```

8. Posicionamento e Layout

display: Controla como um elemento é exibido (block, inline, flex, grid). Essa propriedade é fundamental para o layout.

Exemplo:

```
css
.flex-container { display: flex; }
```

position: Define como um elemento é posicionado na página (static, relative, absolute, fixed, sticky).

Exemplo:

```
css
div { position: relative; top: 10px; }
```

top, right, bottom, left: Usadas em combinação com position para definir a localização de um elemento.

Exemplo:

```
css
div { position: absolute; top: 50px; left: 100px; }
```

float e clear: Controla o posicionamento flutuante de elementos, permitindo que texto e outros elementos contornem elementos flutuantes. clear previne que elementos adjacentes flutuem ao redor.

Exemplo:

```
css
img { float: left; }
p { clear: both; }
```

z-index: Define a ordem de empilhamento de elementos sobrepostos. Um número maior significa que o elemento será exibido na frente.

Exemplo:

```
css
.overlay { position: absolute; z-index: 10; }
```

9. Layout Moderno com Flexbox e Grid

display: flex;: Ativa o modo de layout flexível, permitindo que os elementos filhos sejam distribuídos eficientemente em uma linha ou coluna.

Exemplo:

```
css
.flex-container { display: flex; }
```

justify-content, align-items: Controlam o alinhamento dos itens em um contêiner flexível, tanto no eixo principal (horizontal) quanto no eixo cruzado (vertical).

Exemplo:

```
css
.flex-container {
  display: flex;
  justify-content: space-between;
  align-items: center;
}
```

flex-grow, flex-shrink, flex-basis: Ajusta o espaço que os itens flexíveis ocupam dentro do contêiner.

Exemplo:

```
css
.item { flex-grow: 1; }
```

display: grid;: Ativa o layout de grade CSS, permitindo uma grade bidimensional para colocar elementos em linhas e colunas.

Exemplo:

```
css
```

```
.grid-container { display: grid; grid-template-columns: repeat(3, 1fr); }
```

grid-template-columns, grid-template-rows: Define a estrutura da grade, especificando quantas colunas e suas larguras, bem como quantas linhas e suas alturas.

Exemplo:

```
css
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr); /* Três colunas de igual largura */
    grid-template-rows: 100px 200px; /* Duas linhas com altura específica */
}
```

grid-gap, grid-area, grid-auto-flow: Propriedades que controlam o espaçamento entre os elementos da grade e o fluxo automático dos itens.

Exemplo:

```
css
.grid-container {
    display: grid;
    grid-gap: 10px; /* Espaçamento de 10px entre os elementos */
    grid-auto-flow: dense; /* Preenche os espaços vazios na ordem */
}
```

Subgrid no CSS Grid: Permite que elementos filhos de um grid herdem o layout do contêiner pai, facilitando o alinhamento e espaçamento.

Exemplo:

```
css
.subgrid {
    display: grid;
    grid-template-columns: 1fr 1fr; /* Herança de colunas do elemento pai */
    grid-template-rows: repeat(2, 50px);
}
```

10. Efeitos e Animações

transition: Permite transições suaves entre estados de estilo, definindo duração e tipo de transição.

Exemplo:

```
css
button {
    transition: background-color 0.3s ease;
}

button:hover {
    background-color: blue;
}
```

transform: Aplicado para rotação, escalas e translações de elementos.

Ex.: `rotate(45deg)`, `scale(1.5)`.

Exemplo:

```
css
.box {
    transform: scale(1.2); /* Aumenta o tamanho em 20% */
}
```

animation: Cria animações complexas definidas por keyframes, permitindo transições entre múltiplos estilos.

Exemplo:

```

css
@keyframes example {
    from {background-color: red;}
    to {background-color: yellow;}
}

div {
    animation: example 5s infinite; /* Animação infinita de mudança de
cor */
}

```

box-shadow: Adiciona sombras em elementos, controlando o deslocamento, desfoque e cor da sombra.

Exemplo:

```

css
div {
    box-shadow: 5px 5px 10px rgba(0,0,0,0.5);
}

```

filter: Aplica efeitos visuais como desfoque, brilho e contraste aos elementos.

Exemplo:

```

css
img {
    filter: blur(5px); /* Aplica um desfoque de 5 pixels */
}

```

CSS Motion Path: Permite que elementos se movam ao longo de um caminho específico, criando animações dinâmicas.

Exemplo:

```

css
.animated {
    animation: move 5s infinite alternate;
}

@keyframes move {
    from { transform: translateX(0); }
    to { transform: translateX(100px); }
}

```

11. Design Responsivo e Adaptabilidade

@media: Media Queries permitem aplicar estilos diferentes dependendo das características do dispositivo, como largura de tela.

Exemplo:

```

css
@media (max-width: 600px) {
    body { background-color: lightblue; } /* Estilo para telas
pequenas */
}

```

@container: Permite aplicar estilos baseados nas dimensões do contêiner do elemento, em vez da janela de visualização.

Exemplo:

```

css
@container (min-width: 500px) {
    .responsive-container { background-color: green; }
}

```

CSS Logical Properties: Propriedades que permitem ajustar o layout considerando a direção do texto, útil para idiomas que usam orientações diferentes (ex: Right-to-Left).

Exemplo:

```
css
.container {
    padding-inline-start: 20px; /* Espaçamento à esquerda em LTR e à direita
em RTL */
    margin-block-end: 15px; /* Margem inferior independente da direção do
texto */
}
```

CSS Scroll Snap: Facilita o controle do comportamento de rolagem, permitindo que elementos "se ajustem" em pontos designados ao rolar.

Exemplo:

```
css
.scroll-container {
    scroll-snap-type: x mandatory; /* Ativa o ajuste no eixo X */
    overflow-x: auto;
}

.scroll-item {
    scroll-snap-align: start; /* Alinha o início do item no snap */
}
```

clamp(): Permite criar tamanhos dinâmicos que se ajustam entre um valor mínimo e máximo baseado em um intervalo responsivo.

Exemplo:

```
css
h1 {
    font-size: clamp(1.5rem, 2vw + 1rem, 3rem); /* Tamanho do texto que varia
conforme o viewport */
}
```

12. Metodologias e Arquitetura CSS

BEM (Block, Element, Modifier): Uma metodologia para organizar código CSS, permitindo a criação de classes que refletem a estrutura e o comportamento dos componentes.

Exemplo:

```
html
<div class="menu">
    <a class="menu__item menu__item--active">Home</a>
    <a class="menu__item">Sobre</a>
</div>

/* CSS */
.menu { ... }
.menu__item { ... }
.menu__item--active { ... }
```

OOCSS (Object-Oriented CSS): Foca na separação entre estrutura e aparência, promovendo reusabilidade de estilo.

Exemplo:

```
css
.box { border: 1px solid #ccc; padding: 10px; }

.blue-box { background-color: blue; }
.red-box { background-color: red; }
```

SMACSS (Scalable and Modular Architecture for CSS): Abordagem modular que categoriza estilos, facilitando a manutenção.

Exemplo:

```
css
.layout { ... } /* Estilos de layout */
.module { ... } /* Estilos de módulos */
.state { ... } /* Estilos de estado */
```

Atomic CSS: Utiliza classes utilitárias que podem ser combinadas para compor estilos, como na biblioteca TailwindCSS.

Exemplo:

```
html
<div class="p-4 bg-gray-200 text-center">Olá Mundo</div> <!-- Padding, cor de fundo e alinhamento -->
```

13. Performance e Renderização de CSS

Critical CSS: Carregamento de regras CSS essenciais primeiro, melhorando a performance inicial da página.

Exemplo:

```
html
<style>
/* Apenas os estilos essenciais */
body { background-color: white; }
h1 { color: black; }
</style>
```

Lazy Loading de CSS: Técnica que carrega apenas o CSS necessário para a parte visível da página, utilizando `<link rel="preload">`.

Exemplo:

```
html
<link rel="preload" href="styles.css" as="style"
onload="this.onload=null;this.rel='stylesheet'">
```

CSS Containment: Melhora a performance ao limitar o escopo do que precisa ser re-renderizado quando um elemento muda.

Exemplo:

```
css
.container {
    contain: layout;
}
```

CSS Will-Change: Sinaliza qual propriedade de um elemento provavelmente mudará no futuro, permitindo que o navegador otimize a renderização.

Exemplo:

```
css
.hover-effect {
    will-change: transform;
}
```

14. CSS para Aplicações Complexas

CSS Houdini API: Existem APIs que possibilitam que desenvolvedores alterem comportamentos do CSS usando JavaScript, dando maior controle no layout e no estilo.

Exemplo:

```
javascript
CSS.paintWorklet.addModule('myPaint.js');
```

CSS Paint API: Ferramenta que permite criar desenhos dinâmicos que podem ser usados como imagem de fundo.

Exemplo:

```
javascript
class MyPainter {
    paint(ctx, geom, properties) {
        ctx.fillStyle = 'red';
        ctx.fillRect(0, 0, geom.width, geom.height);
    }
}
registerPaint('my-paint', MyPainter);
```

Shadow DOM e Web Components: Técnicas que encapsulam CSS para que estilos não vazem entre componentes, promovendo modularidade.

Exemplo:

```
html
<style>
  :host {
    display: block;
    border: 1px solid black;
  }
</style>
<script>
  class MyComponent extends HTMLElement {
    constructor() {
      super();
      const shadow = this.attachShadow({ mode: 'open' });
      shadow.appendChild(/* conteúdo aqui */);
    }
  }
  customElements.define('my-component', MyComponent);
</script>
```

Scoped Styles (:scope): Limita a aplicação de estilos dentro de um elemento específico, evitando que eles afetem outros elementos.

Exemplo:

```
css
:scope .highlight { background-color: yellow; }
```

15. Efeitos Visuais Avançados

Blend Modes (mix-blend-mode, background-blend-mode): Usados para misturar cores entre elementos, criando efeitos visuais interessantes.

Exemplo:

```
css
.blender {
  background-color: blue;
  mix-blend-mode: multiply;
}
```

CSS Masks (mask-image, mask-position): Permitem aplicar máscaras em elementos, controlando quais partes do elemento serão visíveis.

Exemplo:

```
css
.masked-element {
  mask-image: url('mask.png');
  mask-size: cover;
```

```
}
```

CSS Shapes (shape-outside): Permitem que o texto flua ao redor de formas não retangulares, dando um efeito visual mais interessante.

Exemplo:

```
css
.shape {
    float: left;
    shape-outside: circle(50%);
    width: 100px;
    height: 100px;
}
```

CSS Multi-Column Layout: Facilita a criação de layouts de várias colunas, útil para publicações e sites de estilo de revista.

Exemplo:

```
css
.multi-column {
    column-count: 3;
    column-gap: 20px;
}
```

16. Novidades e Recursos Emergentes

:has(), :is(), :where(): Novos seletores que permitem combinações mais complexas, melhorando a seleção e aplicação de estilos.

Exemplo:

```
css
div:has(.active) { background-color: yellow; } /* Seleciona divs que
têm uma classe .active */
```

CSS Scrollbar Styling: Permite que desenvolvedores personalizem a aparência das barras de rolagem, melhorando a estética e a usabilidade.

Exemplo:

```
css
::-webkit-scrollbar {
    width: 12px;
}

::-webkit-scrollbar-thumb {
    background: darkgrey;
    border-radius: 6px;
}
```

CSS Exclusions: Permitem que o conteúdo flua ao redor de elementos flutuantes, criando layouts mais dinâmicos.

Exemplo:

```
css
.exclusion {
    float: left;
    shape-outside: polygon(0 0, 100% 0, 0 100%);
}
```

CSS Multi-Column Layout: Facilita a criação de layouts de várias colunas, útil para publicações e sites de estilo de revista.

Exemplo:

```

css
.multi-column {
    column-count: 3; /* Exibe o conteúdo em três colunas */
    column-gap: 20px; /* Espaço de 20px entre as colunas */
}

/* HTML */
<div class="multi-column">
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin at sollicitudin mauris. Sed tincidunt, ipsum eu bibendum fermentum, augue odio finibus ligula, sed tincidunt nunc sapien et dui.</p>
    <p>Curabitur vehicula sem nec fermentum vulputate. Sed vestibulum semper sem ut hendrerit.</p>
    <p>Vestibulum vestibulum nisl ipsum, nec ultrices ligula dictum id. Praesent sit amet.</p>
</div>

```

:has(), :is(), :where(): Novos seletores que permitem combinações mais complexas, melhorando a seleção e aplicação de estilos.

Exemplo:

```

css
div:has(> p) {
    border: 2px solid green; /* Aplica borda se tiver um parágrafo direto dentro do div */
}

.highlight:where(.active, .focus) {
    background-color: yellow; /* Aplica fundo amarelo se a classe for .active ou .focus */
}

```

CSS Scrollbar Styling: Permite que desenvolvedores personalizem a aparência das barras de rolagem, melhorando a estética e a usabilidade.

Exemplo:

```

css
::-webkit-scrollbar {
    width: 12px; /* Largura da barra de rolagem */
}

::-webkit-scrollbar-thumb {
    background: darkgrey; /* Cor do "polegar" da barra de rolagem */
    border-radius: 6px; /* Bordas arredondadas */
}

::-webkit-scrollbar-track {
    background: lightgrey; /* Cor do fundo da barra de rolagem */
}

```

CSS Exclusions: Permitem que o conteúdo flua ao redor de elementos flutuantes, criando layouts mais dinâmicos.

Exemplo:

```

css
.exclusion {
    float: left; /* Ativa a flutuação do elemento */
    shape-outside: circle(50%); /* Cria um contorno circular para o texto fluir ao redor */
    width: 100px; /* Largura da imagem/círculo */
    height: 100px; /* Altura da imagem/círculo */
}

```

JAVASCRIPT

1. Fundamentos do JavaScript

O que é JavaScript?

Uma linguagem de programação usada para desenvolver websites interativos. É executada no navegador, permitindo manipulação de DOM e comunicação assíncrona.

Estrutura de um programa JavaScript

A estrutura básica envolve instruções (códigos), funções, objetos e variáveis que compõem um programa.

Exemplo:

```
javascript
let nome = 'Ana'; // Declaração de variável
function dizerOla() {
    console.log(`Olá, ${nome}!`);
}
dizerOla(); // Chamada da função
```

Ambientes de execução

JavaScript é executado em navegadores, servidores (Node.js) e ambientes de codificação (como consoles).

2. Tipos de Dados

Números

Tipos numéricos que incluem inteiros e flutuantes. Operações básicas como adição, subtração, multiplicação e divisão são aplicáveis.

Exemplo:

```
javascript
let a = 10;
let b = 5.5;
let soma = a + b; // 15.5
```

Strings

Sequências de caracteres. Métodos para manipulação como .length, .toUpperCase(), .toLowerCase(), e concatenação usando +.

Exemplo:

```
javascript
let texto = 'Olá, Mundo!';
console.log(texto.length); // 13
console.log(texto.toUpperCase()); // 'OLÁ, MUNDO!'
```

Booleanos

Valores booleanos true e false. Útil em operações condicionais.

Exemplo:

```
javascript
let isAtivo = true;
if (isAtivo) {
    console.log('O usuário está ativo.');
}
```

Objetos

Estruturas de dados que armazenam coleções de dados e entidades, usando pares de chave–valor.

Exemplo:

```
javascript
const usuario = { nome: 'João', idade: 30 };
console.log(usuario.nome); // 'João'
```

Arrays

Listas ordenadas de valores. Métodos úteis incluem `.push()`, `.pop()`, `.splice()`, e `.forEach()` para iteração.

Exemplo:

```
javascript
const frutas = ['maçã', 'banana', 'laranja'];
frutas.push('uva'); // Adiciona 'uva' no array
console.log(frutas); // ['maçã', 'banana', 'laranja', 'uva']
```

Null e Undefined

`null` é um valor atribuído a uma variável que não tem valor. `undefined` indica que uma variável foi declarada, mas não atribuída um valor.

Exemplo:

```
javascript
let valorNulo = null; // variável sem valor
let valorIndefinido;
console.log(valorNulo); // null
console.log(valorIndefinido); // undefined
```

3. Estruturas de Controle

Condicionais

if/else: Usadas para executar código condicionalmente. A verificação é realizada em expressões booleanas.

Exemplo:

```
javascript
let nota = 7;
if (nota >= 6) {
    console.log('Aprovado!');
} else {
    console.log('Reprovado.');
}
```

switch: Alternativa ao `if/else` para verificar múltiplas condições.

Exemplo:

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

```

switch (dia) {
    case 1:
        console.log('Domingo');
        break;
    case 2:
        console.log('Segunda');
        break;
    case 3:
        console.log('Terça');
        break;
    default:
        console.log('Dia inválido');
}

```

Laços de Repetição (continuação)

while: Repete uma ação enquanto uma condição for verdadeira.

Exemplo:

```

javascript
let contador = 0;
while (contador < 5) {
    console.log(contador); // Imprime números de 0 a 4
    contador++;
}

```

do...while: Similar ao while, mas garante que o código execute pelo menos uma vez.

Exemplo:

```

javascript
let i = 0;
do {
    console.log(i); // Imprime números de 0 a 4
    i++;
} while (i < 5);

```

4. Funções

Funções Declarativas

Declarações de funções que têm um nome e podem ser chamadas por esse nome.

Exemplo:

```

javascript
function apresentar(nome) {
    console.log(`Olá, ${nome}!`);
}
apresentar('Maria'); // 'Olá, Maria!'

```

Expressões de Função

Funções atribuídas a variáveis, que podem ser anônimas ou nomeadas.

Exemplo:

```

javascript
const soma = function(a, b) {
    return a + b;
};
console.log(soma(2, 3)); // 5

```

Funções de Callback

Funções passadas como argumento para outras funções, permitindo execução em momentos específicos.

Exemplo:

```
javascript
function processar(callback) {
    let resultado = 2 + 3;
    callback(resultado);
}
processar(function(res) {
    console.log('Resultado: ' + res); // 'Resultado: 5'
});
```

Arrow Functions

Uma sintaxe mais curta para funções, mantendo o `this` do contexto onde foram definidas.

Exemplo:

```
javascript
const soma = (a, b) => a + b;
console.log(soma(5, 10)); // 15
```

Parâmetros e Argumentos

Parâmetros são variáveis nas declarações de funções, enquanto argumentos são os valores passados a esses parâmetros ao chamar uma função.

Exemplo:

```
javascript
function multiplicar(a, b) {
    return a * b;
}
console.log(multiplicar(4, 5)); // 20
```

Rest e Spread Operators

`...rest`: Coleta múltiplos argumentos em um array. `...spread`: Expande elementos de um array ou objeto.

Exemplo:

```
javascript
function franco(...numeros) {
    return numeros.reduce((a, b) => a + b);
}
console.log(franco(1, 2, 3, 4)); // 10

const partes = ['letra', 'música'];
const banda = ['banda', ...partes]; // ['banda', 'letra',
'música']
console.log(banda); // ['banda', 'letra', 'música']
```

5. Objetos e Arrays

Definição e Manipulação de Objetos

Criação e acesso a propriedades usando notação de ponto e colchetes.

Exemplo:

```
javascript
const carro = {
  modelo: 'Fusca',
  ano: 1970
};
console.log(carro.modelo); // 'Fusca'
```

Métodos de Objetos

Funções definidas dentro de objetos.

Exemplo:

```
javascript
const carro = {
  modelo: 'Fusca',
  andar() {
    return 'Vrum!';
  }
};
console.log(carro.andar()); // 'Vrum!'
```

Manipulação de Arrays: envolve métodos como `push()`, `pop()`, `map()`, `filter()`, `reduce()`, permitindo adicionar, remover e transformar dados.

Exemplo:

```
javascript
// Usando .map() para criar um novo array com quadrados dos números
const numeros = [1, 2, 3, 4, 5];
const quadrados = numeros.map(num => num ** 2); // [1, 4, 9, 16, 25]
console.log(quadrados);

// Usando .filter() para criar um novo array apenas com números pares
const pares = numeros.filter(num => num % 2 === 0); // [2, 4]
console.log(pares);

// Usando .reduce() para somar todos os números
const somaTotal = numeros.reduce((total, num) => total + num, 0); // 15
console.log(somaTotal);

// Criando um array inicial
const numeros = [1, 2, 3, 4, 5];

// Usando .push() para adicionar um número ao final do array
numeros.push(6); // Agora o array é [1, 2, 3, 4, 5, 6]
console.log(numeros);

// Usando .pop() para remover o último número do array
const removido = numeros.pop(); // Remove o 6
console.log(numeros); // Agora é [1, 2, 3, 4, 5]
console.log("Número removido:", removido);
```

6. Programação Assíncrona

SetTimeout e SetInterval: `setTimeout` executa uma função **após um tempo determinado** (em milissegundos), útil para **atrasar execuções**.

`setInterval` executa uma função **repetidamente em intervalos** de tempo, ideal para **loops temporizados**.

Exemplo:

```
javascript
// setTimeout: Executa código após 2 segundos
setTimeout(() => {
    console.log('Executado após 2 segundos');
}, 2000);

// setInterval: Executa código a cada 1 segundo
let contador = 0;
const intervalo = setInterval(() => {
    contador++;
    console.log('Contador:', contador);
    if (contador === 5) {
        clearInterval(intervalo); // Para o intervalo após 5 segundos
    }
}, 1000);
```

Promises são objetos que representam a conclusão (ou falha) de uma operação assíncrona. Elas são usadas para lidar com **tarefas assíncronas**, como requisições a APIs, leitura de arquivos ou timers, evitando o uso excessivo de callbacks.

Exemplo:

```
javascript
const promessa = new Promise((resolve, reject) => {
    const sucesso = true; // Troque para false para ver o erro
    if (sucesso) {
        resolve('Operação realizada com sucesso!');
    } else {
        reject('Erro na operação.');
    }
});

promessa
    .then(resultado => console.log(resultado))
    .catch(erro => console.log(erro));
```

Async/Await é uma sintaxe do JavaScript para trabalhar com funções assíncronas de forma mais legível, evitando o uso excessivo de `.then()`. Ele é usado para esperar a resolução de **Promises**, tornando o código mais limpo e fácil de entender. Sem await, você tentaria pegar o hambúrguer antes de ele ficar pronto. X. Com await, você espera o hambúrguer ficar pronto antes de comer. ✓

Exemplo:

```
javascript
const promessaAssincrona = () => {
    return new Promise((resolve) => {
        setTimeout(() => {
            resolve('Resultado da promessa');
        }, 2000);
    });
};

const executeAsync = async () => {
    console.log('Aguardando resultado...');
    const resultado = await promessaAssincrona();
    console.log(resultado); // 'Resultado da promessa'
};
executeAsync();
```

7. Manipulação de DOM

Selecionando Elementos Isso é feito com métodos como `getElementById()`, `querySelector()` e `getElementsByClassName()`, usados para alterar textos, estilos e eventos dinamicamente.

Exemplo:

```
javascript
const elemento = document.getElementById('meuElemento');
const elementoQuery = document.querySelector('.minhaClasse');
console.log(elemento);
console.log(elementoQuery);

<p class="meutexto">Texto original</p>
<script>
    function mudarCor() {
        let elementos =
document.getElementsByClassName("meutexto");
        for (let i = 0; i < elementos.length; i++) {
            elementos[i].style.color = "red"; // Muda a cor para
vermelho
        }
    }
</script>
```

Alterando Propriedades e Estilos

Exemplo:

```
javascript
const meuElemento = document.getElementById('meuElemento');
meuElemento.textContent = 'Texto alterado!'; // Modifica o texto
meuElemento.style.color = 'blue'; // Altera a cor do texto
meuElemento.style.backgroundColor = 'yellow'; // Muda a cor de fundo
meuElemento.style.fontSize = '20px'; // Altera o tamanho da fonte
meuElemento.style.display = 'none'; // Esconde o elemento
meuElemento.style.width = '200px'; // Define a largura
meuElemento.style.height = '100px'; // Define a altura
meuElemento.style.border = '2px solid red'; // Adiciona uma borda
meuElemento.style.opacity = '0.5'; // Torna o elemento semi-
transparente
meuElemento.style.transform = 'rotate(45deg)'; // Rotaciona o elemento
```

Eventos

Exemplo:

```
javascript
const botao = document.getElementById('meuBotao');
botao.addEventListener('click', () => {
    alert('Botão clicado!');
});
```

8. Ajax e Fetch API Permitem que uma página web **busque e envie dados do servidor sem recarregar a página**, usando `XMLHttpRequest` ou `fetch()`.

Requisições AJAX

Exemplo:

```
javascript
const xhr = new XMLHttpRequest();
xhr.open('GET', 'https://jsonplaceholder.typicode.com/posts', true);
xhr.onload = () => {
  if (xhr.status === 200) {
    console.log(JSON.parse(xhr.responseText)); // Processa a
resposta
  }
};
xhr.send();
```

Fetch API

Exemplo:

```
javascript
fetch('https://jsonplaceholder.typicode.com/posts/1') // Faz a
requisição GET
  .then(response => response.json()) // Converte a resposta para JSON
  .then(data => console.log(data)) // Exibe os dados no console
  .catch(error => console.error('Erro na requisição:', error)); // Trata erros
```

9. Módulos

Import e Export permitem organizar o código em arquivos separados usando `import` e `export`. Isso facilita a reutilização e manutenção do código, importando apenas as partes necessárias em cada arquivo.

Exemplo:

```
javascript
// meuModulo.js
export const minhaFuncao = () => {
  console.log('Função do módulo!');
};

// script.js
import { minhaFuncao } from './meuModulo.js'; //from é de onde pode ser
arquivo ou modulo .
minhaFuncao(); // 'Função do módulo!'
```

10. Programação Orientada a Objetos (POO)

Classes e Objetos Classes são modelos para criar objetos, e objetos são instâncias dessas classes. Eles são usados para estruturar código de forma reutilizável, organizando dados e comportamentos.

Exemplo:

```
javascript
class Animal {
  constructor(nome) {
    this.nome = nome;
  }
  fazerSom() {
```

```

        console.log(` ${this.nome} faz barulho.`);
    }
}

const cachorro = new Animal('Rex');
cachorro.fazerSom(); // 'Rex faz barulho.'

```

Herança permite que uma classe (filha) herde propriedades e métodos de outra classe (pai), reutilizando código. É aplicada para estruturar sistemas modulares e evitar repetição, como em frameworks e bibliotecas.

Exemplo:

```

javascript
class Cachorro extends Animal {
    fazerSom() {
        console.log(` ${this.nome} late.`);
    }
}

const meuCachorro = new Cachorro('Bingo');
meuCachorro.fazerSom(); // 'Bingo late.'

```

Encapsulamento e Polimorfismo: Encapsulamento em JavaScript é o princípio de esconder detalhes internos de um objeto, permitindo acesso apenas por métodos específicos, geralmente usando `private fields` (#campo) ou closures.

Polimorfismo permite que métodos com o mesmo nome tenham diferentes comportamentos em classes derivadas, como em herança com `extends` e métodos sobrescritos (`override`).

Exemplo:

```

javascript
class Carro {
    constructor(modelo) {
        this.modelo = modelo;
    }

    mostrarModelo() {
        console.log(`Modelo: ${this.modelo}`);
    }
}

class Onibus extends Carro {
    mostrarModelo() {
        console.log(`Modelo de ônibus: ${this.modelo}`);
    }
}

const meuOnibus = new Onibus('Volvo');
meuOnibus.mostrarModelo(); // 'Modelo de ônibus: Volvo'

```

Encapsulamento: Protegendo Dados Sensíveis

Encapsulamento significa **esconder informações** dentro da classe, impedindo acesso direto a elas.

```

javascript
Exemplo de Proteção de Senha com Encapsulamento
javascript
CopiarEditar
class Usuario {
    constructor(nome, senha) {
        this.nome = nome;
    }
}

```

```

        let _senha = senha; // Variável privada (usando closure)

        // Método para validar senha
        this.validarSenha = (senhaDigitada) => _senha ===
senhaDigitada;
    }
}

const user = new Usuario("João", "1234");

console.log(user.nome); // João
console.log(user.senha); // ✗ undefined (não pode acessar
diretamente!)
console.log(user.validarSenha("1234")); // ✓ true (senha correta)
console.log(user.validarSenha("0000")); // ✗ false (senha errada)

```

11. Erros e Exceções

Tratamento de Exceções em JavaScript é o processo de capturar e lidar com erros durante a execução do código, evitando que o programa quebre. Ele é feito usando `try...catch`, permitindo ações alternativas ou mensagens de erro mais amigáveis.

Exemplo:

```

javascript
try {
    // Código que pode gerar erro
    const resultado = 10 / 0;
    if (!isFinite(resultado)) throw new Error('Divisão por zero');
} catch (error) {
    console.log('Erro capturado: ' + error.message);
} finally {
    console.log('Execução finalizada.'); // Sempre executado
}

```

Lançar Erros (`throw`) no JavaScript permite criar erros personalizados e interromper a execução do código quando algo inesperado acontece. É útil para validar dados e garantir que um programa não continue com valores inválidos.

Exemplo:

```

javascript
function validarIdade(idade) {
    if (idade < 18) {
        throw new Error('Idade não permitida');
    }
    return true;
}

try {
    validarIdade(16);
} catch (e) {
    console.log(e.message); // 'Idade não permitida'
}

```

12. Ferramentas e Encaminhamento

Depuração em JavaScript é o processo de identificar e corrigir erros no código. Pode ser feita com o `console.log()`

Exemplo:

```
javascript
// Insira um breakpoint aqui no Console do navegador
let x = 5;
let y = 10;
let resultado = x + y; // Inspecione valores aqui
console.log(resultado);
```

Console API do JavaScript permite depurar e registrar mensagens no console do navegador usando métodos como `console.log()`, `console.error()` e `console.warn()`. É útil para testar código, depurar erros e monitorar o comportamento de scripts.

Exemplo:

```
javascript
console.log('Mensagem informativa.');
console.error('Mensagem de erro.');
console.warn('Mensagem de alerta.');
```

13. JavaScript em Node.js

O que é Node.js?

Um ambiente de execução JavaScript no lado do servidor que permite a criação de aplicações web escaláveis.

Módulos em Node.js permitem organizar código em arquivos separados para reutilização e manutenção. Eles são usados para importar/exportar funções, classes e variáveis com `require()` ou `import`.

Exemplo:

```
javascript
// meuModulo.js
module.exports = {
    saudacao: function() {
        return 'Olá do módulo!';
    }
};

// script.js
const meuModulo = require('./meuModulo');
console.log(meuModulo.saudacao()); // 'Olá do módulo!'
```

Criação de Servidores com Node.js e o módulo **Express**, permitindo receber requisições e enviar respostas via HTTP. Esse código cria um **servidor HTTP básico** com Node.js, que escuta na porta **3000** e responde "Olá, mundo!" para qualquer requisição. Ele é útil para **APIs simples, backends e comunicação entre sistemas web**.

Exemplo:

```
javascript
const http = require('http');
```

```

const servidor = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Olá, mundo!');
});

servidor.listen(3000, () => {
    console.log('Servidor rodando na porta 3000');
});

```

14. API do navegador

Local Storage e Session Storage: O **LocalStorage** mantém os dados mesmo após fechar o navegador, enquanto o **Session Storage** apaga os dados ao fechar a aba. Eles são usados para salvar preferências do usuário, tokens de login e configurações sem precisar de um banco de dados.

Exemplo:

```

javascript

// Local Storage
localStorage.setItem('nome', 'João');
const nomeArmazenado = localStorage.getItem('nome');
console.log(nomeArmazenado); // 'João'

// Session Storage
sessionStorage.setItem('sessao', 'atual');
const sessaoArmazenada = sessionStorage.getItem('sessao');
console.log(sessaoArmazenada); // 'atual'

```

WebSockets permitem comunicação bidirecional em tempo real entre o cliente e o servidor sem precisar recarregar a página. São usados em **chats, notificações ao vivo e jogos multiplayer**.

Exemplo:

```

javascript

const socket = new WebSocket('ws://echo.websocket.org/');

socket.onopen = () => {
    console.log('Conexão estabelecida');
    socket.send('Olá, WebSocket!');
};

socket.onmessage = (event) => {
    console.log('Mensagem recebida:', event.data); // 'Olá, WebSocket!'
};

```

Geolocalização: permite obter a posição do usuário através da API `navigator.geolocation`, retornando latitude e longitude. É usada em **mapas, rastreamento de entregas, personalização de conteúdo e serviços baseados em localização**.

Exemplo:

```

javascript

if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition((position) => {
        console.log('Latitude: ' + position.coords.latitude);
        console.log('Longitude: ' + position.coords.longitude);
    }, (erro) => {
        console.log('Erro ao obter localização: ' + erro.message);
    });
}

```

```
    } else {
        console.log('Geolocalização não é suportada neste navegador.');
    }
```

15. CSS e JavaScript

Manipulação de Estilos com JavaScript: permite alterar dinamicamente a aparência dos elementos HTML usando o objeto `style`. Isso é útil para criar interações como mudar cores, tamanhos e visibilidades com eventos.

Exemplo:

```
javascript
const meuElemento = document.getElementById('meuElemento');
meuElemento.style.backgroundColor = 'yellow'; // Muda a cor de fundo
meuElemento.style.fontSize = '20px'; // Altera o tamanho da fonte
```

Interação com Frameworks Front-end: permite criar interfaces dinâmicas e reativas usando bibliotecas como **React**, **Vue** e **Angular**, facilitando a manipulação do DOM e a atualização automática de elementos na tela. Eles melhoram a experiência do usuário ao reduzir recarregamentos de página e otimizar a performance.

Exemplo (React):

```
javascript
import React from 'react';

function App() {
    return <h1>Olá, mundo!</h1>;
}

export default App;
```

16. Tópicos Avançados

Performance e Otimização: envolve técnicas para tornar o código mais rápido e eficiente, reduzindo o tempo de execução e consumo de memória. Aplicações incluem **lazy loading**, **debounce/throttle**, **async processing** e **minificação de arquivos** para melhorar a experiência do usuário.

Exemplo (uso de debounce):

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

window.addEventListener('resize', debounce(() => {
    console.log('Janela redimensionada');
}))
```

```
}, 200));
```

JavaScript e WebAssembly

Exemplo: WebAssembly é mais complexo de se demonstrar sem um ambiente específico, mas você pode compilar um código C/C++ para WebAssembly e importá-lo em um projeto JavaScript.

Programação Funcional

Exemplo:

```
javascript
const numeros = [1, 2, 3, 4, 5];
const quadrados = numeros.map(num => num * num);
console.log(quadrados); // [1, 4, 9, 16, 25]
```

Aspectos de segurança: é um paradigma que trata funções como valores e evita mudanças de estado. Aplicada em **map**, **filter**, **reduce**, torna o código mais modular e reutilizável.

Exemplo: Sanitização de entrada

```
javascript
const usuarioEntrada = "<script>alert('XSS')</script>";
const entradaSegura = usuarioEntrada.replace(/</g,
"&lt;").replace(/>/g, "&gt;");
console.log(entradaSegura); //
'&lt;script&gt;alert('XSS')&lt;/script&gt;'
```

17. Novidades do ECMAScript

Novos recursos do ECMAScript (ES6 e além)

Exemplo (Template Literals):

```
javascript
const nome = 'Carlos';
const saudacao = `Olá, ${nome}! Seja bem-vindo.`;
console.log(saudacao); // 'Olá, Carlos! Seja bem-vindo.'
```

18. Manipulação de Eventos

Eventos Personalizados: permitem criar e disparar eventos específicos da aplicação, usando `CustomEvent`. Isso é útil para comunicação entre componentes e módulos de um sistema.

Exemplo:

```
javascript
const meuEvento = new CustomEvent('meuEvento', { detail: { key: 'value' } });
document.addEventListener('meuEvento', (e) => {
    console.log('Evento personalizado recebido:', e.detail);
});
// Dispara o evento personalizado
document.dispatchEvent(meuEvento);
```

Delegação de Eventos: permite atribuir um evento a um elemento pai, em vez de definir individualmente para cada filho, usando **event bubbling**. Isso melhora o desempenho e facilita a manipulação de elementos dinâmicos.

Exemplo:

```
javascript
const lista = document.getElementById('minhaLista');

lista.addEventListener('click', (event) => {
  const itemClicado = event.target;
  console.log('Item clicado:', itemClicado.textContent);
});
```

19. Trabalhando com APIs

RESTful APIs: permitem a comunicação entre sistemas usando requisições HTTP (GET, POST, PUT, DELETE). São amplamente usadas para integrar front-end e back-end, facilitando o acesso a dados de servidores.

GET → Buscar usuários (GET /users)

POST → Criar um usuário (POST /users)

PUT → Atualizar um usuário (PUT /users/1)

DELETE → Remover um usuário (DELETE /users/1)

Exemplo:

```
javascript
const API_URL = "https://jsonplaceholder.typicode.com/users"; // API de exemplo

// GET - Buscar usuários
fetch(API_URL)
  .then(response => response.json())
  .then(data => console.log("Usuários:", data));

// POST - Criar um usuário
fetch(API_URL, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ name: "Novo Usuário", email: "teste@email.com" })
})
  .then(response => response.json())
  .then(data => console.log("Criado:", data));

// PUT - Atualizar um usuário (id 1)
fetch(` ${API_URL}/1`, {
  method: "PUT",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ name: "Usuário Atualizado" })
})
  .then(response => response.json())
  .then(data => console.log("Atualizado:", data));

// DELETE - Remover um usuário (id 1)
fetch(` ${API_URL}/1`, { method: "DELETE" })
  .then(() => console.log("Usuário deletado"));
```

GraphQL: é uma linguagem de consulta para APIs que permite buscar exatamente os dados necessários, reduzindo requisições desnecessárias. Ele é usado no **JavaScript** para criar APIs mais eficientes, flexíveis e rápidas, geralmente com **Node.js** e **Express**.

Exemplo:

```
javascript
fetch('https://example.com/graphql', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    query: '{ users { id, name } }'
  }),
})
.then(response => response.json())
.then(data => console.log(data));
```

20. Considerações de Desempenho

Minimização e Bundling: são técnicas para **reduzir o tamanho dos arquivos e melhorar o desempenho** da aplicação. A **minimização** remove espaços e comentários, enquanto o **bundling** combina múltiplos arquivos JS em um só, reduzindo requisições ao servidor.

Exemplo:

Este tópico é mais sobre ferramentas. Por exemplo, você poderia usar **Webpack** ou **Parcel** para otimizar seu código. A configuração dessas ferramentas envolveria um arquivo de configuração, que podemos incluir como exemplo:

```
javascript
// webpack.config.js
const path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
  mode: 'production',
};
```

Profiling de Performance: é a análise do tempo de execução do código para identificar gargalos e otimizar o desempenho. Aplicado via **DevTools do navegador**, permite medir **tempo de execução de funções, uso de memória e FPS**, ajudando a melhorar a performance de aplicações web.

Exemplo:

Você pode usar o console do desenvolvedor do navegador para identificar gargalos de desempenho, mas uma abordagem simples dentro do código é:

```
javascript
console.time('Meu Processo');
// Código cujo desempenho você deseja medir
console.timeEnd('Meu Processo'); // Mostra o tempo que levou
para executar
```

21. Testes

Testes Unitários: são testes automatizados que verificam se funções ou módulos individuais do código funcionam corretamente. São aplicados usando frameworks como **Jest** ou **Mocha**, garantindo que pequenas partes do código operem como esperado antes da integração completa.

Exemplo:

```
javascript
// usando Jest
const soma = (a, b) => a + b;

test('soma 1 + 2 deve ser igual a 3', () => {
    expect(soma(1, 2)).toBe(3);
});
```

Testes de Integração: verificam se diferentes partes de um sistema funcionam bem juntas, como a comunicação entre APIs, banco de dados e frontend. São aplicados com ferramentas como **Jest**, **Mocha** e **Supertest** para garantir que os componentes interajam corretamente.

Exemplo (Cypress):

```
javascript
// Teste simples em um site
describe('Meu Teste de Integração', () => {
    it('Visitar a página inicial', () => {
        cy.visit('https://example.com');
        cy.contains('Exemplo');
    });
});
```

22. Acessibilidade

Boas Práticas de Acessibilidade: garantem que usuários com deficiência possam interagir com aplicações web. Isso inclui usar **atributos ARIA**, garantir **navegação por teclado** e evitar manipulações que prejudiquem leitores de tela.

Exemplo:

```
html
<button aria-label="Fechar" onclick="fechar()">X</button>
```

23. Princípios de Design (Design Patterns)

Padrões de Projeto: são soluções reutilizáveis para problemas comuns no desenvolvimento de software, ajudando a escrever código mais organizado, flexível e escalável. Exemplos incluem o **Singleton** (uma única instância global), **Factory** (criação de objetos dinâmicos) e **Observer** (notificação de mudanças entre objetos).

Exemplo (Singleton):

```
javascript
class Singleton {
    constructor() {
        if (!Singleton.instance) {
            Singleton.instance = this;
        }
        return Singleton.instance;
}
```

```
    }
}

const instance1 = new Singleton();
const instance2 = new Singleton();
console.log(instance1 === instance2); // true
```

24. Frameworks e Bibliotecas Populares

React: é uma biblioteca JavaScript para criar interfaces de usuário dinâmicas e eficientes, usando componentes reutilizáveis. Ele é amplamente usado no desenvolvimento web para criar aplicações rápidas e interativas, como sites e sistemas web modernos.

Exemplo:

```
javascript

import React from 'react';
import ReactDOM from 'react-dom';

function App() {
    return <h1>Olá, React!</h1>;
}

ReactDOM.render(<App />, document.getElementById('root'));
```

Vue.js: é um framework JavaScript progressivo para construir interfaces de usuário interativas e dinâmicas. Ele é amplamente usado para criar **aplicações web modernas**, como **dashboards, SPAs (Single Page Applications)** e sistemas interativos.

Exemplo:

```
html

<div id="app">
    {{ mensagem }}
</div>

<script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
<script>
    new Vue({
        el: '#app',
        data: {
            mensagem: 'Olá, Vue.js!'
        }
    });
</script>
```

Angular: é um framework JavaScript para construir aplicações web dinâmicas e escaláveis, focado em **componentização e performance**. Ele é usado para criar **SPAs (Single Page Applications)** com **roteamento, formulários, serviços e integração com APIs**.

Exemplo:

```
typescript

import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-root',
  template: '<h1>Olá, Angular!</h1>',
})
export class AppComponent { }
```

25. JavaScript e Segurança

Melhores Práticas de Segurança: JavaScript pode ser vulnerável a ataques como **XSS (Cross-Site Scripting)** e **CSRF (Cross-Site Request Forgery)**, então é essencial validar entradas e usar medidas de proteção.

Melhores Práticas de Segurança: Sempre sanitize dados do usuário, use **HTTPS**, evite **eval()**, implemente **CSP (Content Security Policy)** e proteja **tokens de autenticação**.

Exemplo:

```
javascript
// Sanitização de entrada
function sanitizarEntrada(entrada) {
  return entrada.replace(/</g, "&lt;").replace(>/g, "&gt;");
}

const entradaDoUsuario = "<script>alert('XSS!')</script>";
const entradaSegura = sanitizarEntrada(entradaDoUsuario);
console.log(entradaSegura); // 
'&lt;script&gt;alert(&#39;XSS!&#39;)&lt;/script&gt;'
```

26. ECMAScript Future Features

Propostas EIPs e Futuras Funcionalidades: **Propostas EIPs (Ethereum Improvement Proposals)** em JavaScript referem-se a melhorias no ecossistema Ethereum que impactam desenvolvedores, como mudanças no **Web3.js** e **Ethers.js** para otimizar contratos inteligentes e interações com blockchain.

Exemplo (Optional Chaining):

```
javascript
const usuario = {
  nome: 'Maria',
  endereço: {
    cidade: 'São Paulo'
  }
};

console.log(usuario.endereço?.cidade); // 'São Paulo'
console.log(usuario.endereço?.estado); // undefined (sem erro, evita TypeError)
```

Exemplo (Nullish Coalescing):

```
javascript
const valor = null;
const resultado = valor ?? 'Valor padrão';
console.log(resultado); // 'Valor padrão'
```