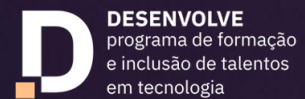



KORU





JS NO BROWSER (DOM)

O QUE VAMOS APRENDER HOJE?

- **Revisão:** JS + HTML/CSS no navegador
- O que é o **DOM** (Document Object Model)
- **Seleção** de elementos
- **Navegação** no DOM
- **Manipulação** (conteúdo, atributos, estilos)
- **Criação, inserção e remoção** de elementos

O QUE VAMOS APRENDER HOJE?

Já vimos o **JavaScript no Node.js** para tarefas de "backend" ou scripts.

Mas o JavaScript "nasceu" para rodar no navegador!

Revisão rápida: Como HTML, CSS e JS trabalham juntos no browser?

HTML: Define a estrutura e o conteúdo da página.

CSS: Define a aparência e o layout.

JavaScript: Adiciona comportamento e interatividade.

Mas como o JavaScript "enxerga" o HTML e o CSS para poder manipulá-los? 🤔



O DOM

DOM

Document Object Model.

Pense no **DOM** como uma representação em árvore da sua página HTML.

Cada elemento HTML (<body>, <h1>, <p>, <div>, etc.) é um nó nessa árvore.

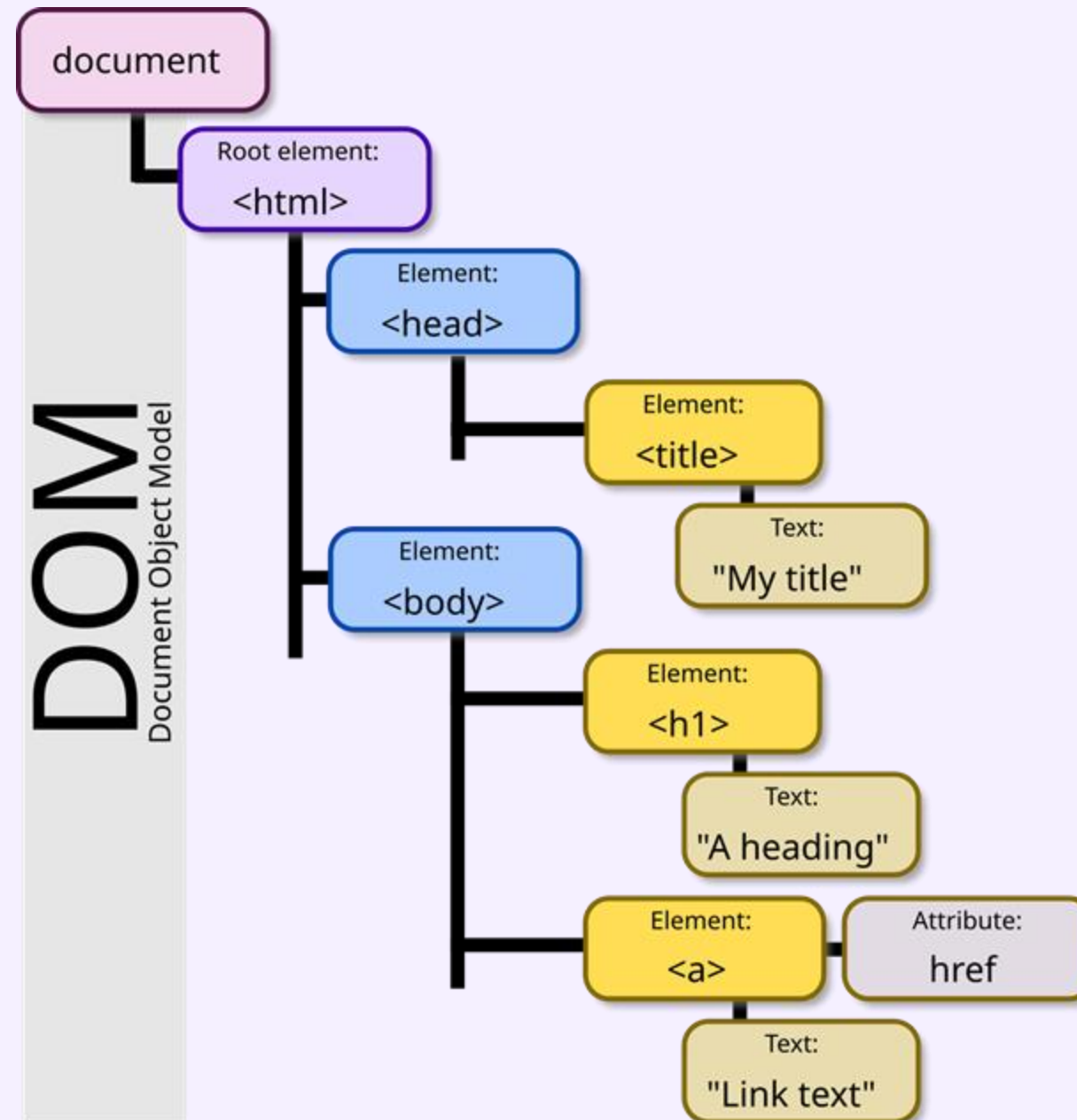
Existem nós para texto, comentários, etc.

O navegador cria essa árvore do DOM quando carrega seu HTML.

O JavaScript usa o DOM como uma interface (API) para acessar e modificar a estrutura, o conteúdo e os estilos da página depois que ela foi carregada.

DOM

https://en.wikipedia.org/wiki/Document_Object_Model



DOM

Visualizando o DOM

A melhor forma de ver o DOM é no seu navegador, usando as Ferramentas do Desenvolvedor (DevTools).

Como abrir:

Clique direito em qualquer lugar da página -> "Inspecionar" ou "Inspecionar Elemento".

Ou use o atalho: F12 (Windows/Linux) ou Cmd + Option + I (macOS).

Na **aba "Elements"** ou "Elementos", você vê a **árvore do DOM**.

Na aba **"Console"**, você pode digitar código JavaScript e interagir diretamente com o DOM da página aberta!



SELECCIONANDO ELEMENTOS

SELECIONANDO ELEMENTOS

O primeiro passo para manipular o DOM é **selecionar** o(s) elemento(s) que você quer modificar.

Existem vários métodos no objeto global document para fazer isso:

document.getElementById(id)

Seleciona UM ÚNICO elemento pelo seu atributo id.

IDs devem ser únicos na página.

```
// Seu JavaScript
const mainTitleElement = document.getElementById('mainTitle');
console.log(mainTitleElement); // Mostra o elemento <h1> no console
```

```
<!-- Seu HTML -->
<h1 id="mainTitle">Olá, Mundo!</h1>
```

SELECIONANDO ELEMENTOS

document.getElementsByClassName(className)

Seleciona TODOS os elementos que possuem uma determinada classe CSS.

Retorna uma HTMLCollection (parecido com um array, mas não é um array "completo").

document.getElementsByTagName(tagName)

Seleciona TODOS os elementos com um determinado nome de tag (ex: 'p', 'div', 'a').

Também retorna uma HTMLCollection.

```
// Seu JavaScript
const textItems = document.getElementsByClassName('text-item');
console.log(textItems); // Mostra uma coleção dos 3 elementos

const allParagraphs = document.getElementsByTagName('p');
console.log(allParagraphs); // Mostra uma coleção dos 2 parágrafos
```

```
<!-- Seu HTML -->
<p class="text-item">Primeiro parágrafo.</p>
<p class="text-item">Segundo parágrafo.</p>
<div class="text-item">Este é um div.</div>
```

SELETORES "MODERNOS"

Estes são mais versáteis e permitem usar qualquer seletor CSS para encontrar elementos.

document.querySelector(cssSelector)

Seleciona o PRIMEIRO elemento que corresponde ao seletor CSS especificado.

document.querySelectorAll(cssSelector)

Seleciona TODOS os elementos que correspondem ao seletor CSS especificado.

Retorna uma NodeList (também parecido com array).

```
// Seu JavaScript

// Seleciona o primeiro elemento com a classe 'post'
const firstPost = document.querySelector('.post');
console.log(firstPost);

// Seleciona o h2 DENTRO do primeiro elemento com a classe 'post'
const firstPostTitle = document.querySelector('.post .post-title');
console.log(firstPostTitle);

// Seleciona TODOS os elementos com a classe 'post-title'
const allPostTitles = document.querySelectorAll('.post-title');
console.log(allPostTitles); // NodeList com 2 elementos

// Seleciona TODOS os links dentro de elementos com a classe 'post'
const allReadMoreLinks = document.querySelectorAll('.post a.read-more');
console.log(allReadMoreLinks); // NodeList com 2 elementos
```

```
<!-- Seu HTML -->
<article class="post">
  <h2 class="post-title">Título do Post</h2>
  <p>Conteúdo...</p>
  <a href="#" class="read-more">Leia Mais</a>
</article>
<article class="post">
  <h2 class="post-title">Outro Post</h2>
  <p>Mais conteúdo...</p>
  <a href="#" class="read-more">Leia Mais</a>
</article>
```




NAVEGANDO PELO DOM

NAVEGANDO NO DOM

Uma vez que você tem um elemento selecionado, pode navegar para outros elementos relacionados a ele na árvore do DOM:


- **element.parentNode:** O pai direto.
- **element.children:** Uma HTMLCollection dos filhos elementos diretos.
- **element.firstElementChild:** O primeiro filho elemento direto.
- **element.lastElementChild:** O último filho elemento direto.
- **element.nextElementSibling:** O próximo elemento irmão (no mesmo nível).
- **element.previousElementSibling:** O elemento irmão anterior.

```
// Seu JavaScript
const h2Element = document.querySelector('h2');

console.log(h2Element.parentNode); // <div id="container">...</div>
console.log(h2Element.previousElementSibling); // <p>Parágrafo 1</p>
console.log(h2Element.nextElementSibling); // <p>Parágrafo 2</p>

const container = document.getElementById('container');
console.log(container.children); // HTMLCollection com <p>, <h2>, <p>
console.log(container.firstElementChild); // <p>Parágrafo 1</p>
```

```
<!-- Seu HTML -->
<div id="container">
  <p>Parágrafo 1</p>
  <h2>Título H2</h2>
  <p>Parágrafo 2</p>
</div>
```

MANIPULANDO CONTEÚDO

MANIPULANDO O CONTEÚDO

Podemos, com o JS, alterar o texto visível em um elemento ou até o HTML dentro dele.

element.textContent:

Obtém ou define apenas o texto contido no elemento e seus descendentes. Ignora tags HTML.

Geralmente mais seguro e rápido se você só quer mudar texto.

element.innerHTML:

Obtém ou define o conteúdo HTML dentro do elemento.

Pode inserir novas tags HTML, mas cuidado com segurança (XSS) se o conteúdo vier de usuários.

```
// Seu JavaScript
const myParagraph = document.getElementById('myParagraph');
myParagraph.textContent = 'Novo texto para o parágrafo.'; // Muda o texto

const myDiv = document.getElementById('myDiv');
myDiv.innerHTML = 'Conteúdo <em>novo</em> com **HTML**!'; // Insere HTML
```

```
<!-- Seu HTML -->
<p id="myParagraph">Texto original.</p>
<div id="myDiv">Conteúdo <strong>antigo</strong>.</div>
```



MANIPULANDO ATRIBUTOS

MANIPULANDO ATRIBUTOS

Atributos são pares nome="valor" nas tags HTML (ex: src, href, class, id). Você pode acessá-los ou modificá-los:

element.getAttribute(attributeName): Obtém o valor de um atributo.

element.setAttribute(attributeName, value): Define (ou adiciona) o valor de um atributo.

element.removeAttribute(attributeName): Remove um atributo.

element.attributeName: Para atributos comuns (como id, className, src, href), você pode acessá-los diretamente como propriedades do objeto elemento.

```
// Seu JavaScript (continuando do HTML anterior)
const myImage = document.getElementById('myImage');
myImage.setAttribute('src', 'new-image.png'); // Muda o src
myImage.alt = 'Nova Imagem'; // Muda o alt (diretamente)

const myLink = document.getElementById('myLink');
const currentHref = myLink.getAttribute('href'); // Obtém o href
console.log(currentHref);
myLink.href = 'new-page.html'; // Muda o href (diretamente)

// Cuidado: element.className substitui TODAS as classes
// myImage.className = 'new-class'; // Remove 'old-class' e adiciona 'new-class'
```

```
<!-- Seu HTML -->

<a id="myLink" href="old-page.html">Link</a>
```

MANIPULANDO ATRIBUTOS - CLASSES

Melhor para Classes: element.classList

element.classList.add('className');

Adiciona uma classe.

element.classList.remove('className');

Remove uma classe.

element.classList.toggle('className');

Adiciona a classe se não existir, remove se existir.

element.classList.contains('className');

Verifica se a classe existe (retorna true/false).

```
const myDiv = document.querySelector('#myDiv');
myDiv.classList.add('active', 'highlight');
myDiv.classList.remove('old');
if (myDiv.classList.contains('active')) {
  console.log('Div está ativa!');
}
```



MANIPULANDO ESTILOS

MANIPULANDO ESTILOS

você pode mudar estilos CSS diretamente via JavaScript.

Use a propriedade **element.style**.

As **propriedades CSS** são escritas em **camelCase** (ex: background-color vira backgroundColor).

Sempre vale analisar se é melhor alterar os estilos diretamente ou através de classes 🤔

```
// Seu JavaScript
const styledDiv = document.getElementById('styledDiv');

styledDiv.style.color = 'blue';
styledDiv.style.backgroundColor = '#f0f0f0';
styledDiv.style.fontSize = '20px';
styledDiv.style.border = '1px solid black';
```

```
<!-- Seu HTML -->
<div id="styledDiv">Esta div será estilizada.</div>
```



CRIANDO ELEMENTOS

CRIANDO ELEMENTOS

Você não está limitado aos elementos que já existem no HTML inicial.

document.createElement(tagName):

Cria um novo nó de elemento do tipo especificado. Ele é criado apenas na memória do navegador (por enquanto).

Após criar, você geralmente define seu conteúdo, atributos, classes, etc., usando as técnicas que já vimos

```
// Manipulando os novos elementos
newParagraph.textContent = 'Este parágrafo foi criado com JavaScript!';
newImage.src = 'path/to/new-image.jpg';
newImage.alt = 'Uma imagem criada dinamicamente';
newListItem.innerHTML = 'Item da lista <strong>dinâmico</strong>';
newListItem.classList.add('dynamic-item');
```

```
// Seu JavaScript
const newParagraph = document.createElement('p');
const newImage = document.createElement('img');
const newListItem = document.createElement('li');

// Mostra o novo elemento <p> (não está na página ainda!)
console.log(newParagraph);
```


ADICIONANDO ELEMENTOS AO DOM (PÁGINA)

Criar um elemento não o coloca na página. Você precisa anexá-lo a um elemento pai existente no DOM.

parentElement.appendChild(childElement): Adiciona childElement como o último filho de parentElement.

parentElement.insertBefore(newElement, referenceElement): Insere newElement como filho de parentElement, antes do referenceElement.

```
// Seu JavaScript
const myList = document.getElementById('myList');
const contentArea = document.getElementById('contentArea');

const newListItem = document.createElement('li');
newListItem.textContent = 'Novo item (append)';
myList.appendChild(newListItem); // Adiciona no final da lista

const anotherItem = document.createElement('li');
anotherItem.textContent = 'Outro item (insertBefore)';
const firstItem = myList.querySelector('li'); // Pega o primeiro LI existente
myList.insertBefore(anotherItem, firstItem); // Adiciona antes do primeiro item

const newHeading = document.createElement('h3');
newHeading.textContent = 'Título Criado!';
contentArea.appendChild(newHeading); // Adiciona no div de conteúdo
```

```
<!-- Seu HTML -->
<ul id="myList">
  <li>Item existente</li>
</ul>
<div id="contentArea"></div>
```



REMOVENDO ELEMENTOS

MANIPULANDO ATRIBUTOS - CLASSES

Assim como você adiciona, você pode remover elementos existentes do DOM.


parentElement.removeChild(childElement):

Remove o childElement especificado do parentElement. Você precisa de referências tanto para o pai quanto para o filho a ser removido.

element.remove(): (Método mais moderno e simples) Remove o elemento diretamente. Você só precisa da referência para o elemento a ser removido.

```
<!-- Seu HTML -->
<div id="parent">
  <p id="paragraphToRemove">Me remova!</p>
  <span>Este span vai ficar.</span>
</div>

<script>
// Seu JS
const paragraphToRemove = document.getElementById('paragraphToRemove');
if (paragraphToRemove) { // Sempre bom verificar se o elemento existe
  paragraphToRemove.remove();
}
</script>
```

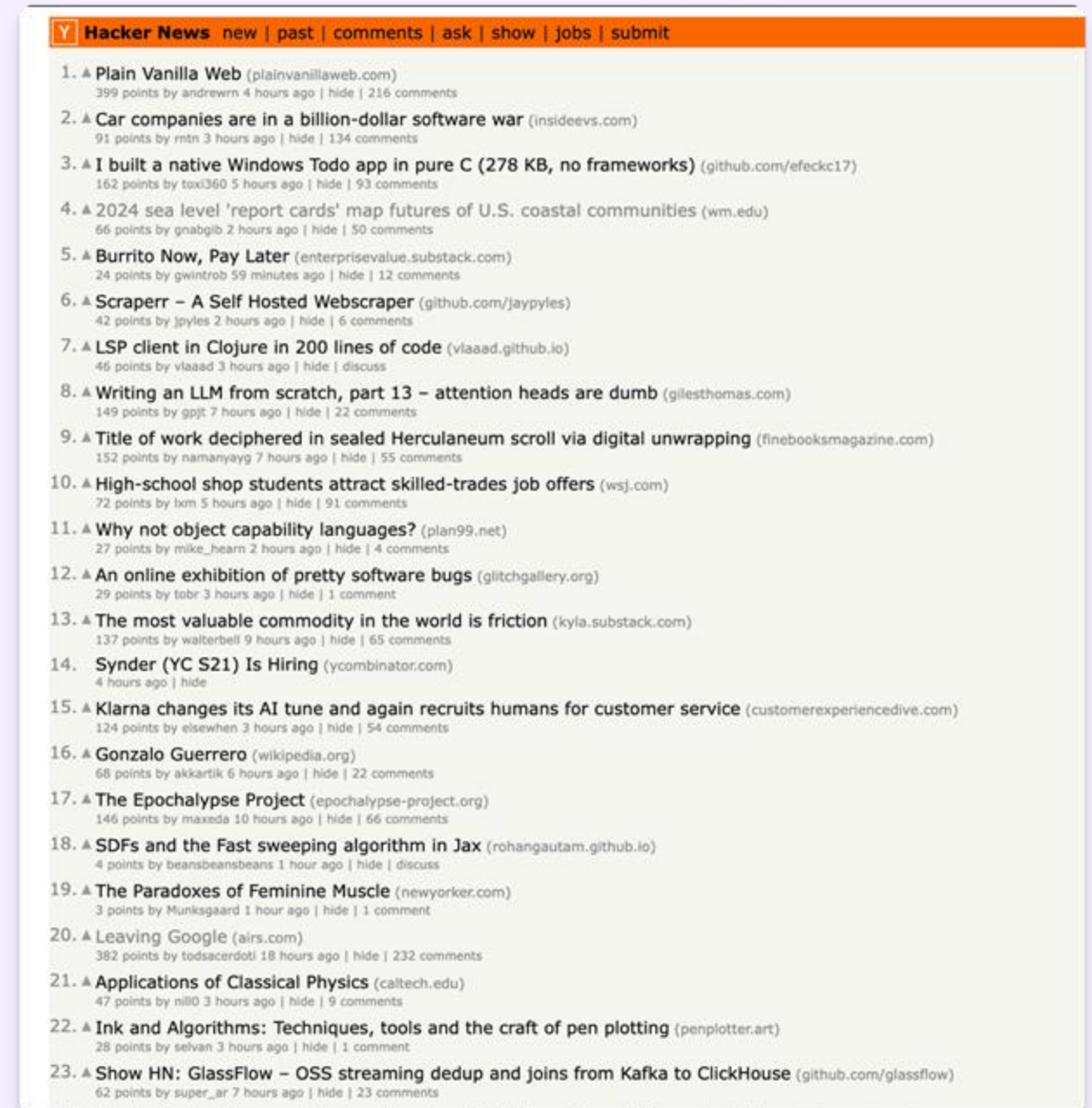
PRÁTICA 01. SCRIPT SCRAPER DE NOTÍCIAS

DESAFIO - SCRIPT SCRAPER DE NOTÍCIAS

Agora que você entende como o JavaScript pode interagir com a estrutura de uma página web através do DOM e sabe selecionar elementos, o desafio é aplicar esse conhecimento de forma prática.

Vamos usar o **site Hacker News como alvo**. Ele tem uma estrutura simples de lista de notícias. Sua tarefa é escrever um pequeno script JavaScript que, rodando no navegador (usando o console das Ferramentas do Desenvolvedor), seja capaz de:

- Selecionar todos os elementos HTML que correspondem aos títulos das notícias na página principal.
- Extrair o texto de cada um desses títulos.
- Imprimir todos os títulos coletados no console do navegador.

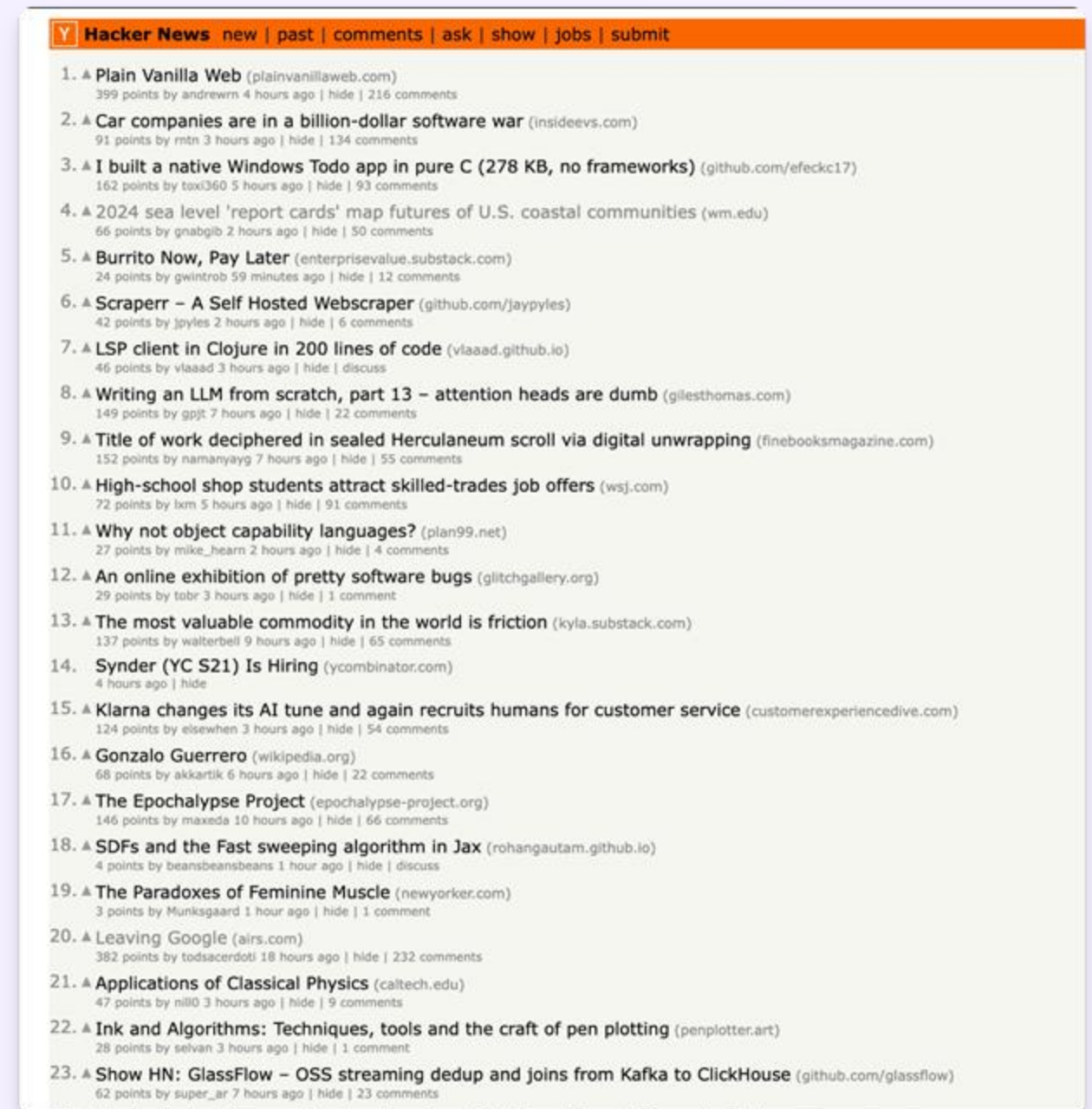


DESAFIO - SCRIPT SCRAPER DE NOTÍCIAS

Passo 1.

Não vamos criar nenhum arquivo HTML ou JS. Vamos fazer tudo isso usando o console do DevTools.

- Use os **métodos de seleção** de elementos que você aprendeu.
- Use a **propriedade correta** para extrair o texto de cada elemento selecionado.



DESAFIO - SCRIPT SCRAPER DE NOTÍCIAS

Passo 2 (desafio adicional).

Está querendo mais aventura? Você conseguiu extrair os títulos no console. Agora, a missão é mais completa:

Crie um website que traz todas as notícias do Hacker News. Estilizado por você!

Para carregar o website você deverá:

Fazer o fetch do hacker news.

Com a resposta, deverá manipular o DOM

Deverá montar seu próprio site com os dados do Hacker News.

Dica: utilize o **DOMParser()** para transformar o resultado de texto do fetch em uma nova árvore DOM.

⚠ ⚠ ⚠ CORS: ⚠ ⚠ ⚠ você provavelmente irá esbarrar com erro de CORS. Uma forma de resolver isso é usando a ferramenta corsproxy.io.

Essa é uma boa oportunidade também para aprender sobre CORS



A group of diverse young people, including men and women of various ethnicities, are smiling and waving. The image is overlaid with a purple gradient. The word "RESUMO" is written in large, white, bold, sans-serif capital letters on the right side of the image.

RESUMO

RESUMO

O **DOM** é a representação em árvore do HTML que o JavaScript usa para interagir com a página.

Usamos o **DevTools** para visualizar o DOM e testar código no Console.

Aprendemos a selecionar elementos por **ID, Classe, Tag e Seletores CSS**

(getElementById, getElementsByClassName, getElementsByTagName, querySelector, querySelectorAll).

Vimos como **navegar entre elementos relacionados** (parentNode, children, nextElementSibling, etc.).

Praticamos **manipular**:

- **Conteúdo**: textContent, innerHTML.
- **Atributos**: getAttribute, setAttribute, removeAttribute, element.property, element.classList.
- **Estilos**: element.style.

Descobrimos como **criar novos elementos** (document.createElement) e **adicioná-los/removê-los da página** (appendChild, insertBefore, removeChild, element.remove()).

KORU

