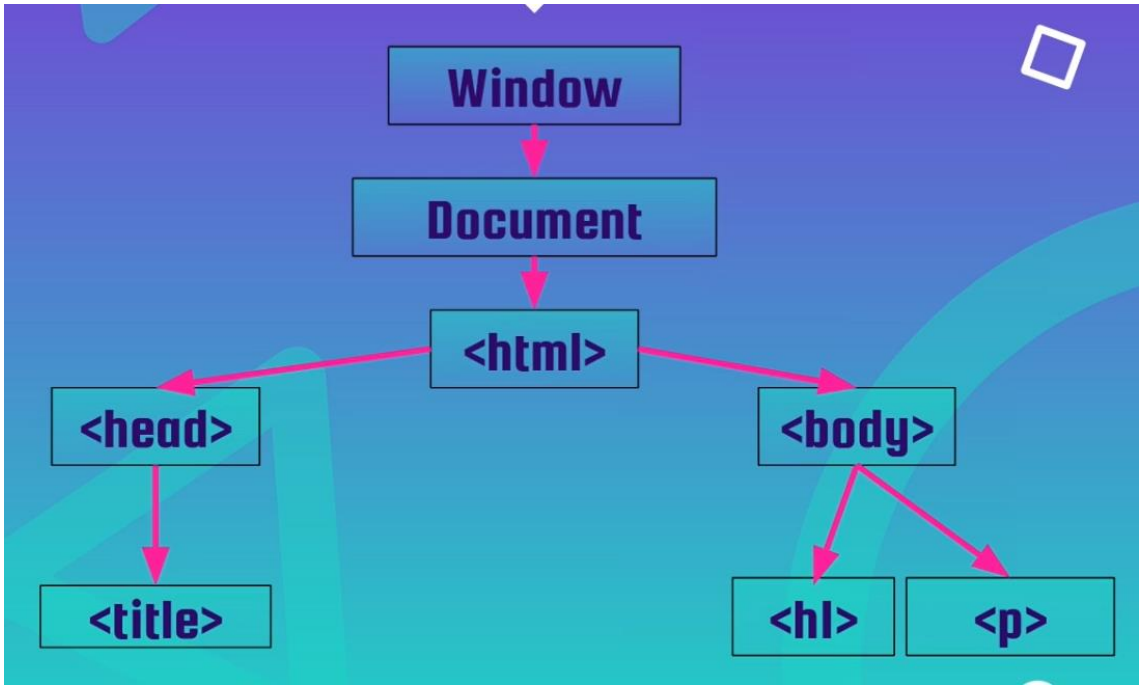
# Cursos de JavaScript na Web – Alura

## Curso 1 – JS na Web Manipule o DOM com JavaScript

1. **Aula 1 – Conhecendo o DOM:**
   1.1. Tudo no DOM é um objeto e todo objeto possui queries. Através delas nós selecionamos as classes ou id das tags para podermos alterar o que quisermos através do JS. Para usar a query nós colocamos o objeto separado por ".", A função seguida de parênteses e dentro dos parênteses colocamos aspas com o indicador da marcação ou a tag em si, e o nome dela dentro para determinarmos qual/quais elementos estamos pegando. Ex.: *document.querySelector('.classe'/'#id'/'tag')*.
      1.1.1. O query sempre irá devolver o primeiro elemento correspondente que encontrar na busca.
   1.2. Podemos também pegar somente o conteúdo de determinado objeto, utilizando a propriedade *textContent*, por exemplo: *document.querySelector('.content').textContent*. Dessa forma ele irá exibir somente o conteúdo, ao invés da tag, sua classe e/ou id e depois o conteúdo.
      1.2.1. Utilizando esta técnica de seleção de elemento, também podemos alterar o conteúdo de uma tag utilizando a mesma linha acima: *document.querySelector('.content').textContent = 'comprar maça'*. Dessa forma, o texto anteriormente colocado na/s tag/s com essa classe será/ão alterados para "comprar maça", independentemente do que estivesse escrito antes.
   1.3. Como a árvore de hierarquia do DOM funciona:




1.4. Outros métodos de seleção de elementos além do querySelector(‘’):

1.4.1. *document.getElementById(‘id’)*: seleciona o elemento pelo id passado.

1.4.2. *document.getElementsByClassName(‘classe’)*: retorna um array dos elementos pelo nome da classe passada.

1.4.3. *document.getElementsByTagName(‘tag’)*: retorna um array dos elementos pelo nome da tag passada

1.4.4. *document.querySelectorAll(seletor)*: devolve todos os seletores com o mesmo nome

## 2. <u>Aula 2 – Comportamento do Formulário:</u>

2.1. O JS é sempre colocado por último no html devido a ordem de carregamento ser de cima para baixo, pois dessa forma, caso tenha algum problema no nosso código, evita que o carregamento da página seja interrompido como seria se ele tivesse sido colocado por último, mantendo o cliente dentro do nosso site.

2.2. Podemos importar um arquivo JS colocando um *src=”localDoArquivo”* dentro da tag <*script*>. Não precisamos escrever o código juntamente com o html.

2.3. Um formulário tem suas próprias propriedades e uma delas é recarregar a página quando o botão é pressionado, mesmo que não aconteça nada.

2.4. Para fazer com que um item seja adicionado a uma lista após escrevê-lo no input text de um forms, precisamos mexer no botão pela árvore DOM usando JS, ao invés do input.

2.5. Existe um quarto seletor que podemos utilizar para selecionar itens específicos por querySelector no JS chamado de: *"data-atributes"*. Basta colocar a palavra *"data-"* e o nome que deseja dar àquele elemento no html.

2.5.1. Ele serve como um separador de responsabilidade.

2.5.2. A principal vantagem é separar o que é do css e o que é do JS, pois se utilizarmos as classes outra pessoa que trabalha no mesmo projeto pode acabar mudando a classe por achar que não está semântica para o CSS e acabar quebrando o código. Utilizando os *data*'s esse risco é bem menor.

2.5.3. Para se referir ao *data* colocado no html, no querySelector, precisamos colocar "[]" dentro das '' e, dentro dele, o nome exato do *data* utilizado para se referenciar àquele elemento. Ex.: *document.querySelector('[data-nome-atribuído-ao-elemento]')*.

2.6. Podemos colocar qualquer expressão que será utilizada várias vezes dentro de uma variável constante.

2.6.1. No caso do projeto referente à este primeiro curso, a variável é o button, então podemos colocar: *const novaTarefa = document.querySelector('[data-form-button]')*.

2.6.2. Desse modo, sempre que quisermos utilizar essa expressão toda dentro do nosso código, podemos colocar a variável que não muda "novaTarefa". Isso é válido para qualquer expressão.

2.7. Para fazer com que o botão faça uma ação quando for clicado precisamos adicionar um Evento de monitoramento à ele, para que sempre que for clicado, execute uma função específica. Traduzindo isso para o JS fica literalmente assim: *novaTarefa.addEventListener('click', ()=>{console.log('fui clicado')})*, respectivamente. Ou seja, vc adicionou um evento de ouvir para o botão que agora está com o nome de novaTarefa e disse para ele que quando ele "ouvir" que o botão foi clicado, irá executar a função anônima que é imprimir a frase "fui clicado" no console do browser.

2.7.1. O problema é que ele irá aparecer e sumir muito rápido, para preservar o log do console você precisa clicar nas settings dele e marcar a opção "preserve log". Dessa forma ele não desaparece.

2.8. Podemos pegar somente o valor de um input, ou seja, seu texto, utilizando o *nomeDaVariável/expressãoDoInput.value*, dessa forma pegando apenas o que estiver escrito no input, ao invés de toda sua expressão html.

2.8.1. Para mandar ele imprimir no console o texto do input, podemos colocar essa expressão dentro da função anônima do event listener, que ocorre quando apertamos o botão.

2.8.2. Lembrando que podemos criar variáveis para todas essas expressões, facilitando o entendimento do código.

2.9. Nós conseguimos fazer com que o formulário pare de atualizar a página, ou seja, pare de mandar informações para o servidor colocando um *preventDefault()*, dentro do eventListener.

2.9.1. Primeiro precisamos nomear aquela função anônima que irá imprimir o texto do input no log e em seguida, dentro da função, colocamos o nome dela seguido de um ponto e a sentença acima.

2.9.2. Fazendo isso, assim que clicarmos no botão o formulário não irá atualizar a página como estava fazendo antes.

2.10. Para organizar melhor o nosso código, nós podemos criar uma função para não deixar tudo bagunçado no eventListener.

2.10.1. Para criar uma função precisamos de uma constante com o nome da função recebendo o evento entre parênteses e todos os parâmetros que estavam dentro do eventListener.

2.10.2. Agora podemos colocar apenas o nome da função depois da vírgula no eventListener ao invés de toda a função que estava previamente.

2.11. Mais informações sobre *data-atributes*: https://cursos.alura.com.br/data-attributes-do-html5-c109

2.12. O que aprendemos na aula de hoje:

2.12.1. Utilizar data-attributes;
2.12.2. Utilizar o método addEventListner para escutar eventos no elemento;
2.12.3. Prevenir o comportamento padrão do formulário.

## 3. Aula 3 – Adicionar Item na Lista:

3.1. Para adicionarmos conteúdo no html de forma dinâmica nós utilizamos o parâmetro *InnerHTML* seguido da variável que queremos alterar, recebendo o conteúdo alterado.

3.1.1. Isso não adiciona um novo conteúdo, apenas substitui o anterior dependendo das variáveis e elementos selecionados.

3.2. Para criarmos conteúdo/tags dentro do DOM/HTML, precisamos utilizar a propriedade *document.createElemet('tag')*.

3.3. Como existe hierarquia de parentesco nas tags HTML, precisamos colocar o parâmetro *tagMãe.appendChild(tagFilho)*. Ao fazer isso o novo conteúdo/tag será incorporado ao HTML dinamicamente.

3.3.1. Porém, ao fazer isso o texto não vem com a formatação CSS, será apenas um texto incorporado ao corpo do HTML sem nenhuma estilização.

3.3.2. Para contornar esse problema, adicionamos a classe do css que já existe para esse elemento utilizando o comando: *variávelDoElemento.classList.add('classeCSS')*.

3.3.3. Fazendo isso todos os novos elementos referentes a essa variável criados dinamicamente irão apresentar a classe escolhida, que já estará configurada no CSS.

3.3.4. Depois disso tudo, o conteúdo adicionado dinamicamente estará totalmente estilizado da maneira correta.

3.3.5. O *appendChild* sempre cria um novo elemento no final no do nó, portanto, sempre que um novo item for criado ele será alocado logo após o anterior.

3.4. Vídeo da Alura explicando melhor sobre template strings: https://cursos.alura.com.br/template-string-c123

3.5. Todos os elementos na nossa árvore do DOM são nós e todos os nós podem ser acessados via JavaScript. Os nós podem ser deletados, criados ou modificados. Durante o curso utilizamos o método *appendChild* que sempre é adicionado no final do nó, para colocar um nó filho dentro do nó mãe. Outros métodos para manipular nós:

3.5.1. *insertBefore(pai, filho)*: Coloca um nó antes do outro;

3.5.2. *replaceChild( elemento1, elemento2)*: Substitui o nó elemento 1 pelo nó elemento2;

3.5.3. *removeChild(elemento)*: Remove um nó da árvore.

3.6. O que aprendemos na aula:

3.6.1. Utilizar template strings;

3.6.2. Colocar um elemento filho dentro do elemento pai utilizando o método *appendChild*;

3.6.3. Criar elementos utilizando o método *createElement*.

## 4. Aula 4 – Concluir Tarefa:

4.1. Utilizando todos os métodos vistos até agora nós conseguimos criar e inserir botões em qualquer lugar de um HTML.

4.2. Para alterarmos um estilo precisamos adicionar o comando *toggle* logo após a propriedade *classList* de um elemento. Dentro das '' dentro dos () você coloca qual a propriedade CSS que deseja que aconteça.

4.3. O *toggle* sempre devolve um booleano, ou seja, um valor verdadeiro ou falso.

4.4. Um grande problema do JS é você deixar seu código no escopo global, pois dessa forma qualquer pessoa tem acesso ao seu código e suas funções de finidas.

4.4.1. A solução para esse problema é criar uma função anônima externa ao código original, pois dessa forma, qualquer coisa que a pessoa digitar no console do browser aparecerá como indefinido.
4.4.2. Para fazer isso basta colocar todo o código entre () e dentro de uma função anônima, ou seja: (() => { CódigoCompleto }). Dessa forma o nosso código fica protegido.
4.4.3. Porém, ao tentar executar o código dessa forma veremos que ele não irá funcionar. Para corrigir isso colocamos () no final de tudo, ficando assim: (() => { CódigoCompleto })(). Desse modo e com essas alterações, o código funcionará normalmente e ficará protegido.
4.4.4. Essa técnica se chama IIFE (*Immediately Invoked Function Expression ou Função de Invocação Imediata*).

4.5. O que aprendemos na aula de hoje:
4.5.1. Adicionar classe CSS utilizando o método toggle;
4.5.2. Utilizar o atributo parentElement para subir um elemento na árvore do DOM;
4.5.3. Encontrar o alvo do evento utilizando a propriedade target;
4.5.4. Utilizar IIFE.

## 5. <u>Aula 5 – Remover Tarefa:</u>

5.1. Sempre que criamos uma função de um componente, o nome começa com letra maiúscula.
5.2. Para remover um objeto/elemento utilizamos a propriedade *.remove()* no JS.
5.3. Módulos são pequenas partes do código que formam um todo.
5.4. Para criar módulos precisamos exportar o código de um arquivo que é o módulo e importar nos arquivos que queremos utilizar esse módulo.
5.4.1. Para exportar usamos: *export default nomeDaFunção/Elemento*.
5.4.2. Para importar usamos: *import nomeDaFunção/Elemento from "./localização"*
5.4.3. Para exportar mais de um objeto do módulo colocamos: *export{ objeto1, objeto2 }*
5.5. Para dizer para o nosso código que ele é um módulo, precisamos ir no HTML onde o script está sendo importado e colocar uma propriedade *type="module"* confirmando esse fato.
5.6. O que aprendemos na aula de hoje:
5.6.1. Utilizar import/export;
5.6.2. Remover elementos do DOM com o método remove;
5.6.3. Entender Same Origin Police e CORS.