Incorporando JavaScript em HTML

Existem quatro maneiras de incorporar Javascript em HTML. Vamos abordar as três primeiras, por ser mais comum:

- Entre as tags <script></script> dentro do código HTML.
- A partir de um arquivo externo.
- Em atributos de tratamento de eventos HTML.
- Em uma URL que use o protocolo especial javascript

O elemento <script>

O código JavaScript pode estar dentro do código HTML, entre as marcações **<script></script>**. Observe a sintaxe abaixo:

```
<body>
    <script>

        // Seu código JavaScript aqui
    </script>
</body>
```

Script em arquivo externo

A marcação <script> suporta um atributo src, que especifica a URL de um arquivo contendo código JavaScript. Por convenção, os arquivos JavaScript têm nomes que terminam com a extensão .js.

A sintaxe é:

```
 <script src="caminho e nome do arquivo"></script>
```

Interação com o usuário

Para exibir uma caixa de alerta, utilize o método/ função alert(). o código abaixo entre as tags <script></script> no código de exemplo criado.

Exemplo 01:

```
<script>
        var nome = "Conhecendo o JavaScript";
        alert("Este é o conteúdo da variável nome: " + nome );

    </script>
```

Para incluir botões “OK” e “Cancelar” nas mensagens, utilize o método **Confirm()**. Ele retorna true se for clicado “OK” e false, caso contrário.

Exemplo 02:

```
<script>
var pergunta = confirm("Você está gostando do JavaScript?\n Se SIM clique
OK, Senão clique CANCELAR");
if(pergunta == true) {alert("Ficamos imensamente felizes que você esteja
gostando!");
} else {
  alert("Poxa! Que pena, temos certeza que vai melhorar?");
}
</script>
```

Para interagir com o usuário, tem outro método interessante: a função prompt(). Com ela, é possível solicitar informações ao usuário.

Exemplo 03:

```
<script>
    var nomeUsuario = prompt("Qual o seu nome?");
    if(nomeUsuario !="") {alert("Seu nome é: "+nomeUsuario);
    } else {alert("Nenhum nome foi digitado...");
    }
 </script>
```

Podemos apenas utilizar a variável, que vai valer-se do princípio da lógica booleana onde vazio é falso e tendo conteúdo verdadeiro. O mesmo exemplo ficaria assim:

```
<script>
    var nomeUsuario = prompt("Qual o seu nome?");
    if(nomeUsuario) {alert("Seu nome é: "+nomeUsuario);
    } else {alert("Nenhum nome foi digitado...");
    }
 </script>
```

Temporizador

O método setTimeout() agenda a execução de uma função, decorrido o tempo especificado em milissegundos.

Exemplo 04:

```
<script>
      setTimeout('bemVindo()',5000);
      function bemVindo(){
        alert("Seja bem vindo!");
    }
    </script>
```

Além, do setTimeout(), há outro método setInterval(). Ele funciona de forma bem parecida com setTimeout(). A diferença é que ele irá repetir o código a cada intervalo de tempo informado, ou até que o método clearIterval(), seja executado.

Exemplo 05:

```
<script>
     var cont = 0;
     var t = setInterval('tempo()',5000); // no intervalo de tempo executa a
função tempo()
     function tempo() { // a função tempo incrementa a variável cont de 1 em
1 até 5
      cont++;
      alert("O contador está em: "+cont);
      if(cont == 5){
        alert("Fim da contagem!");
        clearInterval(t); //interrompe o intervalo
        }
    }
    </script>
```

Em resumo, esse código cria um comportamento de contagem progressiva, onde o valor da variável **cont** aumenta a cada 5 segundos até atingir 5, momento em que a contagem é interrompida.

Manipulando Janela

Com método **Open()** e evento **onclick**

O método Open() pode ser usado para abrir uma nova janela do navegador.

Exemplo 06:

```
<h1>abre em outra janela o google.com.br</h1>
    <input type="button" value="acessar google" onclick="abrirJanela();"
/>
    <hr>
    <script>
        function abrirJanela(){open("http://www.google.com.br");
     }
    </script>
```

O exemplo acima vai criar um botão na janela, e ao ser clicado vai efetuar o evento onclick, e executar a função abrirJanela(), que abre em nova janela o google.com.br

Eventos

Quando falamos de eventos em Javascript, nos referimos a determinados acontecimentos dentro do navegador. Vejamos alguns exemplos comuns de eventos em Javascript:

Evento	descrição
onclick	Quando o usuário clica em algum elemento da página.
onchange	Quando um elemento HTML muda. É muito comum utilizar esse evento para controlar o preenchimento de campos de um formulário.
onmouseover	Quando o usuário passa o cursor sobre algum elemento.
onmouseout	Quando o cursor que estava sobre algum elemento, sai.
onkeydown	Quando alguma tecla é pressionada.

Nós podemos definir funções para serem invocadas sempre que um desses eventos forem registrados pelo navegador. Isso nos leva a uma situação muito interessante, pois podemos criar páginas interativas, que vão responder a ações do usuário.

Alguns exemplos :

onmouseover

```
<div id="quadrado">
    <p onmouseover="principal()">mover o mouse para cá</p>
    </div>
```

Este evento é disparado quando o mouse passa sobre a região determinada, a execução da função ocorre automaticamente após a passagem do mouse no local indicado.

Onclick

```
<h1 onclick="digitar()">clicar aqui</h1>
    <div id="msg">
    <p id=mensagem></p>
    </div>
```

Este evento já foi utilizado antes, a função só é executada após o clique no local determinado..

onkeydown

O evento onkeydown pode ser aplicado a um elemento específico. Faria sentido, por exemplo, aplicá-lo a um campo de formulário, desta maneira quando o usuário digitasse alguma coisa dentro deste campo, o evento seria registrado. Mas este evento pode ser aplicado também ao objeto document Escreva o código abaixo, recarregue a página e experimente apertar qualquer tecla.

```
<body>
    <div id="output"></div>
    <script>
        let output = document.getElementById("output");
        document.onkeydown = function() {
        output.innerHTML="Você apertou alguma tecla";
    };
    </script>

</body>
```

Combinando o evento onkeydown com método close()

O método close() é o contrário do método open() enquanto este abre uma página, o método close fecha a página atual.

No exemplo abaixo, quando uma tecla é pressionada a página fecha automaticamente, não importando qual tecla foi pressionada.

```
<script>
        let output = document.getElementById("output");
        document.onkeydown = function() {
        window.close()}
    </script>
```

onchange

O evento onchange() é um método utilizado muitas vezes para testar o preenchimento ou alteração de dados de um formulário. Não é o método mais prático, porém pode ser utilizado para essa finalidade.

Mas enfim o que faz o evento?

Quando ocorre alguma mudança, exemplo: quando clica fora de um input text, é similar ao evento onmouseover e onmouseout, porém esse atua quando é perceptível mudanças na caixa.

Por exemplo podemos utilizar para deixar o conteúdo todo em maiúsculo, vejamos o exemplo abaixo:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>usando eventos no Javascript</title>

</head>
<body>
    Insira seu nome: <input type="text" id="nome" onchange="minhafuncao()">
    <p>Ao clicar fora do input text o texto escrito ficará maiúsculo.</p>
    <script>
        function minhafuncao(){
            var x=document.getElementById("nome");
            x.value=x.value.toUpperCase();
        }
    </script>
</body>
</html>
```

Combinando eventos e funções com passagem de parâmetros

No segundo exemplo temos a utilização de funções e eventos, porém os eventos são executados mediante a parâmetros passados para uma variável, nesse exemplo utilizamos uma variável **acao**, para receber as o evento a ser executado pelo navegador.

Neste exemplo haverá duas ações possíveis, uma delas é ação de alterar o conteúdo da caixa de input, a segunda ação será clicar no botão do formulário.

Observando que quando uma ação é executada a outra não nem carregada em memória, então não é executada, o programa executa apenas uma função por vez, e isso é determinado pelo parâmetro que foi repassado a função.

Observe a utilização do IF, isso nos prende a apenas uma possibilidade distinta da outra.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Controle de textos</title>
    <script>
        function executarAcao(acao){
            if(acao === 'change'){
                alert ("Houve alteração no texto!");
            }else if(acao === 'click'){
                alert ("você clicou!");
            }
        }
    </script>
</head>
<body>
<p><input type="text" onchange="executarAcao('change');" value="Modifique o
texto aqui"></p>
<p><input type="button" onclick="executarAcao('click');" value="Clique
aqui"></p>
</body>
</html>
```

Combinando os eventos onmouseover e onmouseout

Esses eventos são responsáveis pelas ações que o usuário faz com o mouse, passando o mouse no elemento e ou tirando o mouse do elemento, respectivamente.

Como funcionam os eventos?

Onmouseover

O evento onmouseover executa um JavaScript quando o ponteiro do mouse é movido para um elemento ou para um de seus filhos.

onmouseout

Já o evento onmouseout executa um JavaScript quando o ponteiro do mouse é movido para fora de um elemento ou de seus filhos.

Exemplo:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Eventos com mouse</title>
</head>
<body>
    <div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:
20px;padding:40px;color:white;">Esta com dúvidas? passa o mouse aqui!</div>
    <script>
        function mOver(obj){
            obj.innerHTML="Consulte o manual..."
        }
        function mOut (obj){
            obj.innerHTML="Duvidas???"
        }
    </script>

</body>
</html>
```

LocalStorage

Local Storage é uma propriedade do objeto window que pode armazenar dados no computador do usuário e servem para lembrar suas informações, sempre que ele visitar o site. Podemos guardar o seu nome, e-mail, suas preferências, histórico de utilização do site ou qualquer outra informação. (Passwords e informações sensíveis nunca devem ser guardados em local storage).

Informações devem ser armazenadas em local storage como duplas de chave e valor.

Os dados armazenados ficam no histórico do navegador até que seja eliminado do storage.

Podemos fazer isso através de funções que limpam o storage, mas não é considerado uma boa prática porque pode ser de interesse do usuário manter os dados no storage.

Exemplo em sala:

```
<!DOCTYPE html>
<html>
<head>
  <title>Armazenar Nome e E-mail</title>
  <style>
    #mensagem {
      color:red;
      display: none;
    }
  </style>
  <script>
    function armazenarDados() {
      var nome = document.getElementById('nome').value;
      var email = document.getElementById('email').value;
      localStorage.setItem('nomeArmazenado', nome);
      localStorage.setItem('emailArmazenado', email);
      exibirMensagem(nome, email);
      esconderFormulario();
    }

    function exibirMensagem(nome, email) {
      var mensagem = document.getElementById('mensagem');
      mensagem.innerHTML = 'Bem-vindo(a), ' + nome + '! <br>Seu e-mail é: '
+ email;
      mensagem.style.display = 'block';
    }
```

```
    function esconderFormulario() {
      var formulario = document.getElementById('formulario');
      formulario.style.display = 'none';
    }

    // Verificar se o nome e o e-mail já foram armazenados no localStorage
    window.onload = function() {
      var nomeArmazenado = localStorage.getItem('nomeArmazenado');
      var emailArmazenado = localStorage.getItem('emailArmazenado');
      if (nomeArmazenado && emailArmazenado) {
        exibirMensagem(nomeArmazenado, emailArmazenado);
        esconderFormulario();
      }
    };
  </script>
</head>
<body>
  <h1>Armazenar Nome e E-mail</h1>

  <div id="formulario">
    <label for="nome">Digite seu nome:</label>
    <input type="text" id="nome"><br>

    <label for="email">Digite seu e-mail:</label>
    <input type="email" id="email"><br>

    <button onclick="armazenarDados()">Armazenar</button>
  </div>

  <h2 id="mensagem"></h2>
</body>
</html>
```

window.onload - o evento onload dispara um determinado evento, no caso do exemplo acima inicia uma função implícita: **function().**

Especialistas em JavaScript utilizam muito pouco o window.onload, atribuem a isso o tempo de inicio do evento, considerado mais lento do que outras alternativas, bem como utilizar o onload no inicio da página dentro da tag BODY

<body onload="funcao()">

O evento onload utilizado para iniciar uma função na body também podemos utilizar também o método document.onload para iniciar funções fora da tag body
Trazendo para o documento, através do comando **document**

As diferenças entre eles.

A diferença entre document.onload e window.onload é: **document.onload** aciona antes do carregamento de imagens e outros conteúdos externos**.** Ele é acionado antes do **window.onload.** Enquanto o **window.onload** é acionado quando a página inteira é carregada, incluindo arquivos CSS, arquivos de script, imagens, etc.

Comando Document (algumas considerações).

O comando document é muito utilizado no JavaScript, pois, é através dele que podemos fazer modificações no HTML, essas possibilidades de modificações é o que chamamos de DOM.

Como o document funciona no JavaScript

O comando document é um objeto, e nele, contém várias propriedades que são funções para fazermos alguma coisa no nosso DOM. Esses sub-comandos do document são chamados de seletores de NÓS (node).

Veja alguns comandos por exemplo:

Como selecionar uma div em JavaScript

```
//Selecionando uma div
document.querySelector('div.classe_da_minha_div')
```

Isso fará com que o JavaScript resgate um NÓ do nosso DOM e retorne para você. Logo, você pode criar uma variável chamada div e colocar esse comando nela (se você não conhece sobre as variáveis no JavaScript

```
//selecionando uma div
let div = document.querySelector('div.classe_da_minha_div')
```

Esse sub-comando do document pode ser usado para qualquer tag do HTML, você só precisa referênciar ela dentro dos parênteses e das aspas, por exemplo, vamos supor que eu quero selecionar um input com o name cep:

```
//Selecionando um input com name cep
let cep = document.querySelector('input[name="cep"]')
```

Perceba que dentro do querySelector, nós informamos a tag HTML que desejamos e depois algum atributo para especificarmos.
O atributo não é necessário, se colocarmos apenas input, ele irá selecionar o primeiro input que estiver no DOM (no nosso HTML).