JavaScript

Aula 3

JS

Funções

Uma função JavaScript é definida com a palavra chave **function**, seguida por um **nome**, seguido por **parênteses ()** .

Os nomes das funções podem conter letras, dígitos, sublinhados e cifrões (mesmas regras das variáveis).

Os parênteses podem incluir nomes de parâmetros separados por vírgulas:

(parâmetro1, parâmetro2, ...)

O código a ser executado, pela função, é colocado entre **chaves: {}**

JS

Invocação/chamando uma Função

O código dentro da função será executado quando "algo" **invocar** (chamar) a função:

- Quando ocorre um evento (quando um usuário clica em um botão);
- Quando é invocado (chamado) a partir do código JavaScript; e
- Automaticamente (auto-invocado).

JS

Exemplo de uma função simples:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function carroFunction()
      {
        let carNome = "Ferrari";
        document.write(carNome);
      }
    </script>
  </head>

  <body>
    <h2>Função</h2>
    <p>Executando função alocada na tag head</p>
    <script>
      carroFunction();
    </script>
  </body>
</html>
```

Atividade Prática 1

1. Fazer um programa em HTML para:
2. Criar uma função na tag <head> para atribuir valores para três variáveis;
    a) nome;
    b) telefone;
    c) bairro;
    i. A função deve retornar as informações concatenadas;
3. Criar um título com a tag <h2>;
4. Criar um parágrafo com informações pertinentes ao que a função se propõe;
5. Chamar a função criada para exibir as informações (tag <body>;

Atividade Prática 1

Gabarito:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function pessoaFunction()
      {
        let nome = "Pedro Nascimento";
        let telefone = 977651245;
        let bairro = "Jardim América";
        document.write("O nome é: " + nome + " cujo telefone é: " + telefone + " e mora no bairro: " + bairro);
      }
    </script>
  </head>
  <body>
    <h2>Função Pessoa</h2>
    <p style="color: cornflowerblue">Executando função pessoa alocada na tag head</p>
    <script>
      pessoaFunction();
    </script>
  </body>
</html>
```

JS

Retorno de Função

Quando o JavaScript atinge uma instrução **return**, a função para de ser executada.

Se a função foi chamada a partir de uma instrução, o JavaScript "retornará" para executar o código após a instrução de chamada.

As funções geralmente calculam um **valor de retorno**. O valor de retorno é "retornado" de volta ao "chamador"

JS

Exemplo de uma função com return:

```
<html>
  <head>
        <script>
              function multiplica(p1, p2)
                    {
                          return p1 * p2;
                    }
        </script>
  </head>
  <body>
        <h2>Função</h2>
        <p>Executando uma função para multiplicar números</p>
        <script>
              document.write (multiplica(4, 3));
        </script>
  </body>
</html>
```

Atividade Prática 2

1. Fazer um programa em HTML para:
2. Criar uma função na tag <head> para calcular a média de duas notas (as notas são os parâmetros). A função precisa usar o return para devolver a média calculada;
3. Criar um título com a tag <h2>;
4. Criar um parágrafo com informações pertinentes ao que a função se propõe;
5. Informar as duas notas através do prompt;
6. Chamar a função criada e atribuir o resultado a uma variável (tag <body>);
7. Na sequência é preciso saber se a media é maior ou igual a 7, se for exibir aprovado, se não exibir reprovado.

JS

Eventos

Representam toda a força motriz que permite toda a dinâmica da página.

Os eventos fazem parte do **DOM** e podem ser entendidos como **“gatilhos”** que são disparados sempre que algo acontecer, como um clique do mouse sobre um elemento, a movimentação do mouse, alguma tecla pressionada no teclado ou uma ação de abrir ou fechar uma página.

O **DOM** fica sempre observando tudo o que está acontecendo na página, no documento e em seus conteúdos. Assim que houver alguma interação do usuário, o **DOM irá disparar o gatilho**.

JS

Os eventos mais comuns são:

- **onclick**: disparado quando o usuário clica sobre o elemento.
- **onchange**: disparado quando um elemento HTML foi alterado.
- **onmouseover**: disparado quando o usuário passa o mouse “por cima” do elemento.
- **onmouseout**: disparado quando o usuário movimenta o mouse “para fora” do elemento.
- **onkeydown**: disparado quando o usuário pressiona alguma tecla do teclado.
- **onload**: disparado quando o navegador termina de carregar a página.

JS

Exemplo de uma função com uso de evento:

```
<html>
        <head>
                <title>Funcoes</title>
                <script>
                        function calcsoma(num1,num2)
                        {
                                let tot;
                                tot = num1+num2;
                                alert("A soma foi, " + tot);
                        }
                </script>
        </head>
        <body>
                <form>
                        <h1 style = "font-family: fantasy; color: red">Agora faremos o cálculo</h1>
                        <Button onclick = "calcsoma(3, 8)">Clique aqui para calcular</Button>
                </form>
        </body>
</html>
```

Atividade Prática 3

1. Criar um arquivo .html;
2. Criar uma função na tag <head> para somar dois números;
3. Exibir o resultado do somatório com o comando alert;
4. Criar na tag <body> a entrada de dados de dois valores utilizando o prompt;
5. Criar um título com a tag <h2>;
6. Criar um botação para chamar a função;

Atividade Prática 3

Gabarito com o uso do form (pegando valores do input):

```
<html>
	<head>
		<title>Funcoes</title>
		<script>
			function calcsoma(num1,num2)
			{
				let tot;
				tot = num1+num2;
				alert("A soma foi, " + tot);
			}
		</script>
	</head>
	<body>
		<form>
			<h1 style = "font-family: fantasy; color: cornflowerblue">Agora faremos o informe de valores</h1>
			<label for="var1">Digite o primeiro numero</label><br>
			<input type="text" name="var1"> <br><br>
			<label for="var2">Digite o segundo numero</label><br>
			<input type="text" name="var2">
			<br>
			<br>
			<Button style="color: cadetblue" onclick = "calcsoma(parseInt(var1.value),
			parseInt(var2.value))">Clique aqui para calcular</Button>
		</form>
	</body>
</html>
```

Atividade Prática 4

1. Criar um arquivo .html;
2. Criar uma função na tag <head> para somar dois números;
3. Criar uma função na tag <head> para subtrair dois números;
4. Criar uma função na tag <head> para multiplicar dois números;
5. Exibir o resultado do somatório com o comando alert. Um em cada função;
6. Criar na tag <body> a entrada de dados de dois valores utilizando o prompt;
7. Criar um título com a tag <h2>;
8. Criar um botão para chamar a função soma;
9. Criar um botão para chamar a função subtrair;
10. Criar um botão para chamar a função multiplicar;

Atividade Prática 4

Gabarito:

```
<html>
<head>
    <title>Funcoes</title>
    <script>
      function calcsoma(val1, val2)
      {
        let tot;
        tot = val1 + val2;
        alert("o valor total da soma e, " + tot);
      }
      function calcsub(val1, val2)
      {
        let tot;
        tot = val1 - val2;
        alert("o valor total da subtracao e, " + tot);
      }
      function calcmult(val1, val2)
      {
        let tot;
        tot = val1 * val2;
        alert("o valor total da multiplicacao e, " + tot);
      }
    </script>
</head>
<body>
      <form>
        <label for="var1">Digite o primeiro numero</label><br>
        <input type="text" name="var1"> <br><br>
        <label for="var2">Digite o segundo numero</label><br>
        <input type="text" name="var2">
        <h2 style="color: darkcyan">Agora faremos o informe dos resultados</h2>
        <Button onclick = "calcsoma(parseInt(var1.value), parseInt(var2.value))">Clique aqui para somar</Button>
        <Button onclick = "calcsub(parseInt(var1.value), parseInt(var2.value))">Clique aqui para subtrair</Button>
        <Button onclick = "calcmult(parseInt(var1.value), parseInt(var2.value))">Clique aqui para multiplicar</Button>
      </form>
</body>
</html>
```

JS

innerHTML

A propriedade innerHTML define ou retorna o conteúdo HTML (HTML interno) de um elemento.

A maneira mais fácil de obter o conteúdo de um elemento é usando a propriedade innerHTML.

A propriedade innerHTML é útil para obter ou substituir o conteúdo de elementos HTML.

JS

Exemplo do evento onclick:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Usando o evento onclick</title>
        </head>
        <body>
                <h1 onclick="innerHTML='Isso acontece quando usamos o evento
                onclick!'">Clique nesse link para testar o evento onclick!</h1>
        </body>
</html>
```

JS

Exemplo do evento onmousedown e onmouseup:

```
<!DOCTYPE html>
<html>
		<head>
			<title>Eventos onmousedown e onmouseup</title>
		</head>
<body>
		<div onmousedown="mDown(this)" onmouseup="mUp(this)"
		Style="background-color:#D94A38;width:90px;height:20px;padding:40px;">Clique aqui</div>

		<script>
			function mDown(obj)
				{
				obj.style.backgroundColor="#1ec5e5";
				obj.innerHTML="Solte o clique"
				}

			function mUp(obj)
				{
				obj.style.backgroundColor="#D94A38";
				obj.innerHTML="Obrigado"
				}
		</script>

</body>
</html>
```

JS

Document Object Model - DOM

Quando uma página é carregada, o navegador cria um **Document Object Model** da página.

O navegador interpreta cada palavra, letra ou símbolo de um HTML e exibe o resultado na tela:

- Esse resultado é uma página visível para o usuário do navegador

Resultado da interpretação do HTML é armazenado em uma estrutura de objetos:

- Document Object Model (DOM)

JS

Cada elemento, atributo e texto HTML no DOM torna-se um objeto

- Objetos podem ser acessados de modo independente pelos scripts.

É um padrão definido pelo W3C para acesso a documentos

DOM do HTML define:

- Objetos e propriedades de todos os elementos em um documento HTML/XHTML;
- Métodos para manipular (obter/modificar/adicionar/apagar) cada elemento.

JS

Todo documento XHTML é uma hierarquia de elementos:

- Todos são subordinados à *tag<html>;*
- <html> possui um cabeçalho e um corpo (body); e
- Corpo possui outras tags ligadas a ele.

Essa hierarquia, em DOM, é representada como uma estrutura de dados de árvore.

DOM define objetos e propriedades para cada elemento:

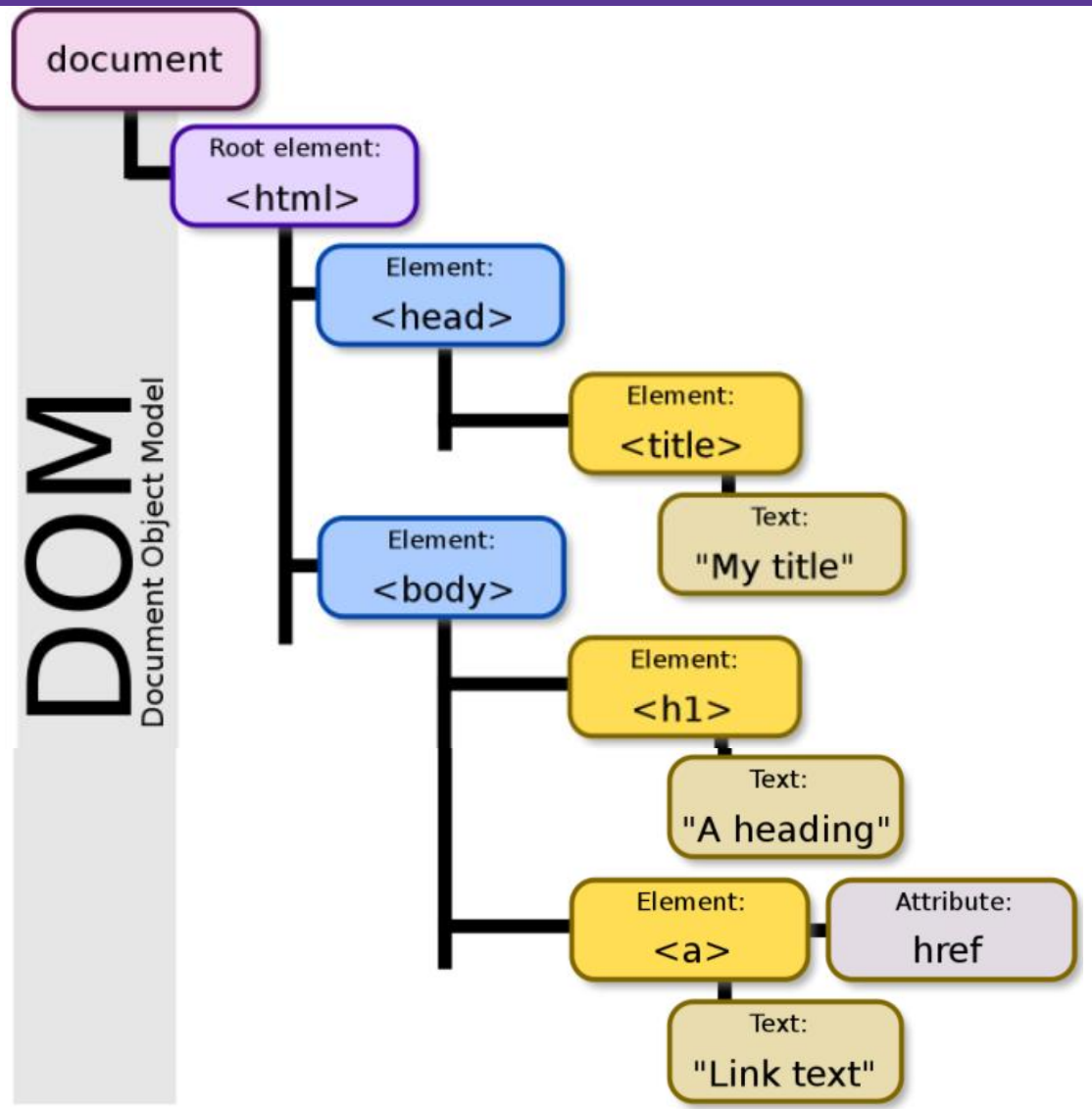
- Cada objeto de elemento é ligado ao seu objeto “pai” na árvore DOM

JS

Com o modelo de objeto, JavaScript tem todo o poder necessário para criar HTML dinamicamente:

- JavaScript pode alterar todos os elementos HTML na página;
- JavaScript pode alterar todos os atributos dos elementos HTML na página;
- JavaScript pode alterar todos os estilos CSS;
- JavaScript pode remover um elemento HMTL e seus atributos;
- JavaScript pode adicionar um elemento HMTL e seus atributos;
- JavaScript pode reagir a todos os eventos que ocorrerem em uma página; e
- JavaScript pode criar novos eventos na page.

JS


DOM
Document Object Model
document
Root element:
<html>
Element:
<head>
Element:
<title>
Text:
"My title"
Element:
<body>
Element:
<h1>
Text:
"A heading"
Element:
<a>
Attribute:
href
Text:
"Link text"

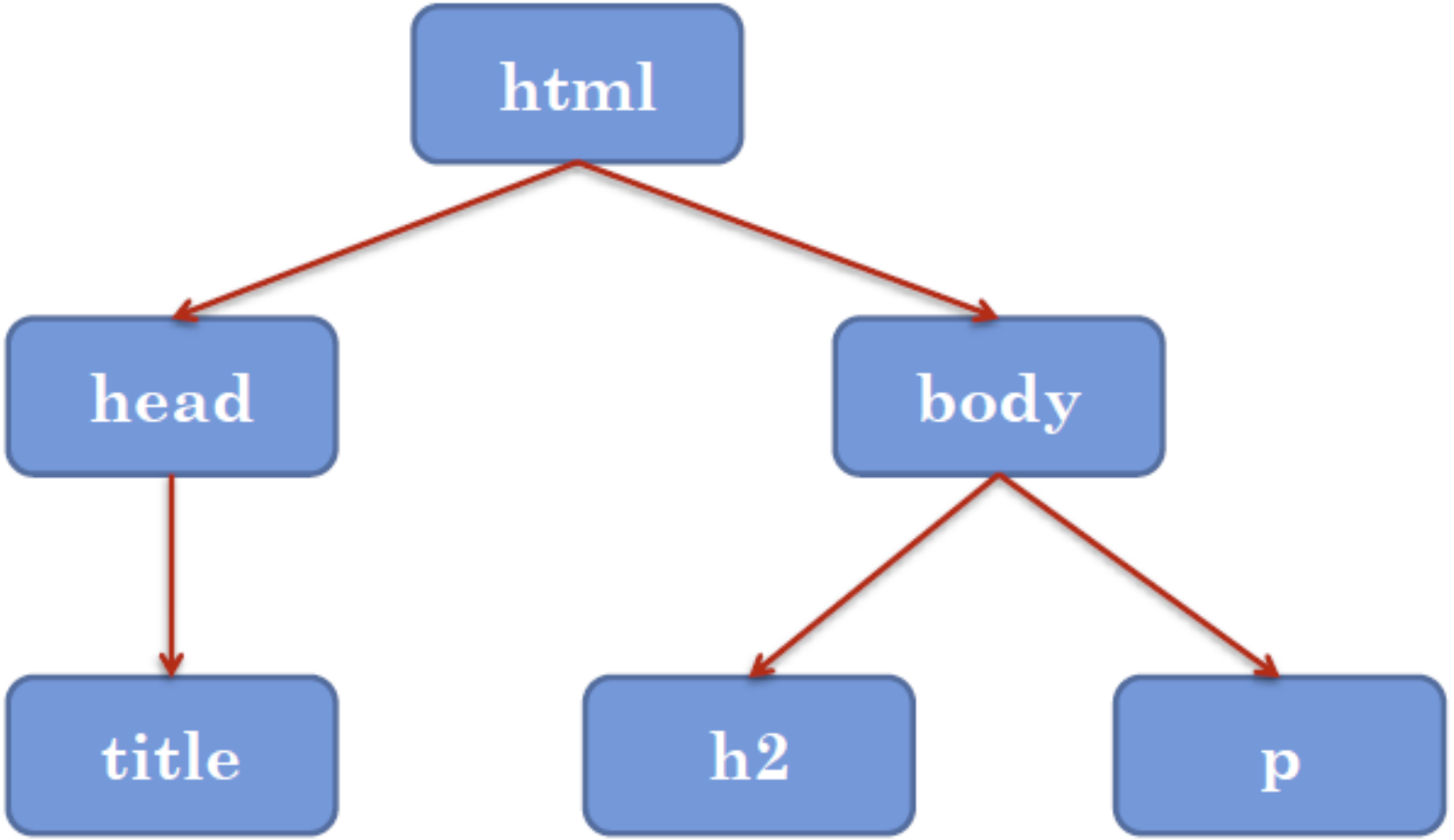
JS

EXEMPLO:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Testando DOM</title>
        </head>
        <body>
                <h2>DocumentObjectModel(DOM)</h2>
                <p>DOM permite o acesso e a manipulação de
                documentos HTML por meio de funções
                acessíveis ao JavaScript.
                </p>
        </body>
</html>
```

JS

Representação do exemplo anterior na árvore DOM



JS

NÓS da árvore DOM

Cada elemento do HTML é representado como um “nó” na árvore DOM.

Nó de documento (document):

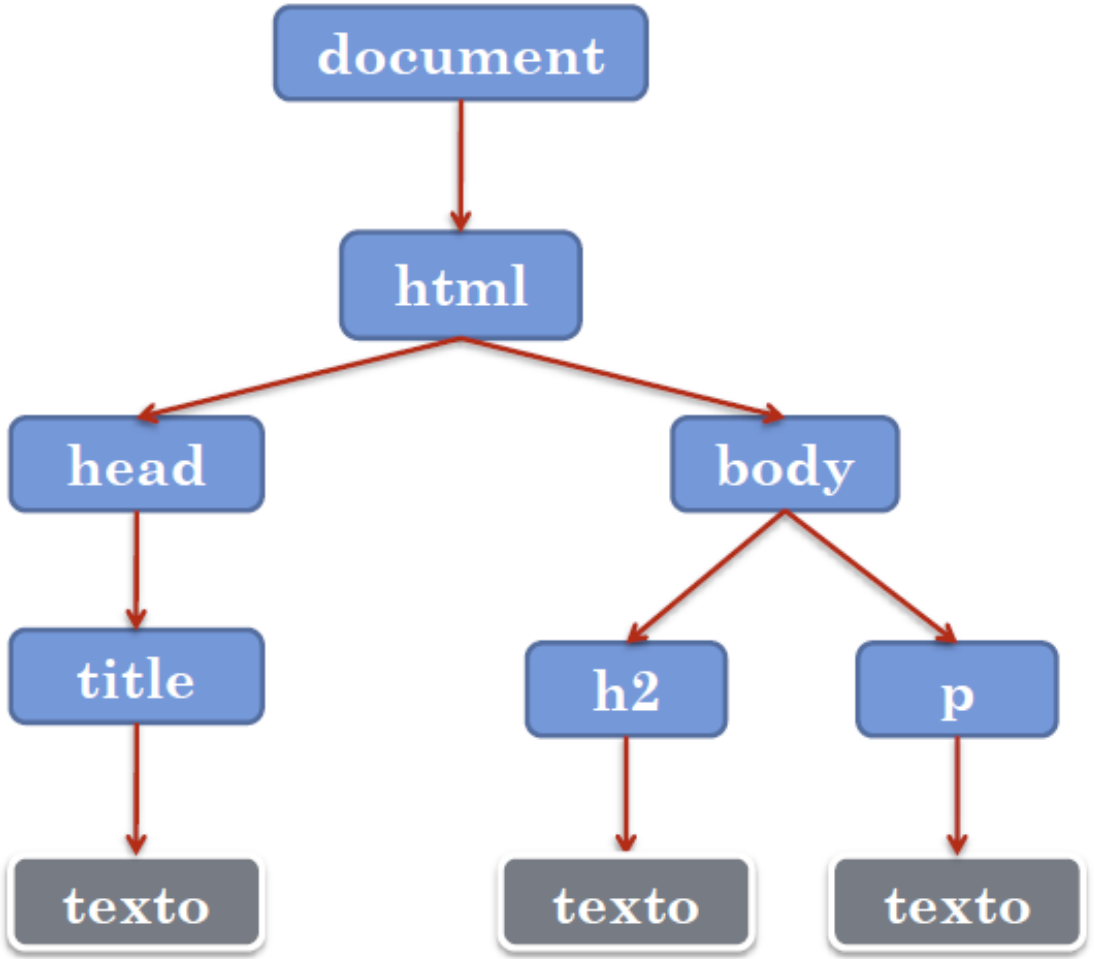
- É a raiz de toda a representação DOM; e
- Ponto de entrada para obter os outros nós.

Tipos de nós:

- Elemento *<a>*
- Atributo *href, id, class, etc.*
- Texto “*http://www.abc.com*”
- Comentário *<!--texto ->*

JS

Exemplo anterior com informações complementares



JS

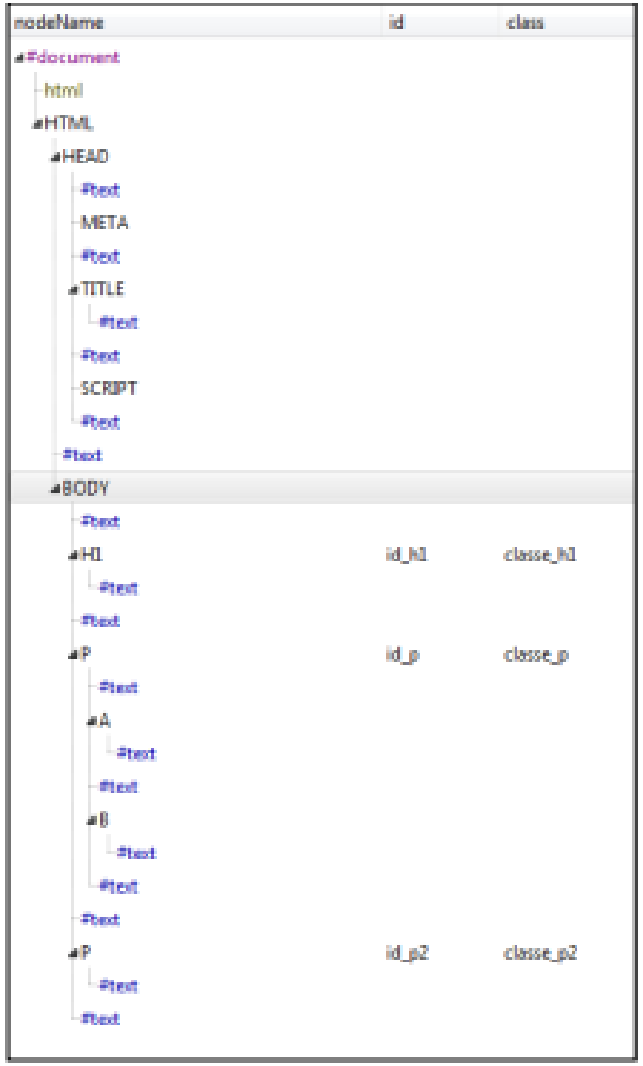
Relacionamento entre os nós: pais, filhos e irmãos

Em uma árvore:

- Nó topo é chamado de **principal** (root);
- Todo nó possui um **pai** (exceto o root);
- Um nó pode ter qualquer número de **filhos**;
- **Irmãos** (siblings) são nós com o mesmo pai; e
- Um nó sem filhos é chamado de **folha**.

JS


#document
Pai de todos
HTML
HEAD
BODY
Filhos de BODY e Irmãos
META
SCRIPT
TITLE
H1
P
P
#text
#text
#text
#text
Filhos de HEAD e Irmãos
A
B
#text
#text
Filhos de P e Irmãos


nodeName
id
class
#document
html
HTML
HEAD
#text
META
#text
TITLE
#text
#text
SCRIPT
#text
#text
BODY
#text
H1
id_h1
classe_h1
#text
#text
P
id_p
classe_p
#text
A
#text
#text
B
#text
#text
#text
P
id_p2
classe_p2
#text
#text

JS

Existe um outro objeto que é pai de todos estes elementos (não representado na estrutura), que representa a página do navegador: o objeto **window**.

Com o uso do objeto **window**, temos acesso à vários recursos do navegador, como, por exemplo:

- Abrir uma nova janela no navegador (window.open);
- Fechar uma janela (window.close);
- Definir que site será acessado (window.location);
- Definir o tamanho e posicionamento das janelas (propriedades width, height, top e left de window.open); e
- Exibir ou ocultar itens da janela, como barra de endereços, barra de status, barras de rolagens, entre outros.

JS

O objeto **document**, como podemos imaginar, representa o documento HTML em si e toda sua estrutura. Com ele, por meio de seus métodos e atributos, podemos:

- acessar um elemento específico, pelo seu id, nome, tag ou classe (apenas por id retorna um único elemento, os demais podem retornam um array de elementos).
    - document.getElementById();
    - document.getElementsByName()
    - document.getElementsByTagName()
    - document.getElementsByClassName()
- Acessar o título do documento (document.title);
- acessar a URL do documento (document.URL); e
- Criar um elemento HTML pelo JavaScript (document.createElement), entre outros.

JS

O objeto **element**, por sua vez, representa os elementos html, como **<body>, , <p>** etc. Com esse objeto, podemos acessar o elemento e:

- Modificar sua classe CSS **(elemento.className)**;
- Modificar seu conteúdo **(elemento.innerHTML e document.outerHTML)**;
- Saber quem são seus elementos filhos **(document.children)**; e
- Ter acesso a seus atributos **(document.attributes)**, entre outros.

JS

Graças ao DOM, podemos ter uma padronização de comportamentos nos diferentes navegadores. **Seus métodos e atributos deram liberdade de programação para tornar as páginas ainda mais dinâmicas, usando o JavaScript**. Logo, outros frameworks criaram seus métodos para facilitar o uso do DOM, automatizando alguns processos e criando uma linguagem para usar o DOM de uma forma mais simples. Um dos primeiros frameworks a fazer isso foi o **JQuery**, em uso até hoje.

Veja na sequência uma comparação na forma de modificação de um conteúdo com o **DOM** puro, em seguida com o **JQuery**, que usa “atalhos” para executar os comandos DOM por trás:

JS

Document Object Model – DOM

Enquanto objeto, possui **métodos (funções) e propriedades (atributos)**:

Funções já conhecidas do objeto document:

- getElementById;
- write; e
- addEventListener.

Propriedades já conhecidas do objeto document:

- innerHTML;
- Value; e
- Style.

JS

Encontrando elementos

Propriedade	**Descrição**
document.getElementById(*id*)	Encontra um elemento pelo id
document.getElementsByTagName(*name*)	Encontra um elemento pelo nome da tag
document.getElementsByClassName(*name*)	Encontra um elemento pelo nome da classe

Método	**Descrição**
element.setAttribute*(attribute, value)*	Altera o valor do atributo de um element HTML

JS

innerHTML

A propriedade innerHTML define ou retorna o conteúdo HTML (HTML interno) de um elemento.

A maneira mais fácil de obter o conteúdo de um elemento é usando a propriedade innerHTML.

A propriedade innerHTML é útil para obter ou substituir o conteúdo de elementos HTML.

Exemplo com evento onclick:

```
<html>
        <body>
                <button onclick="innerHTML=Date()">Veja a hora
                </button>
        </body>
</html>
```

JS

Exemplo:

O valor de um nó de atributo é uma referência ao próprio atributo:

- Como um apontador para o atributo.

O texto dentro de um atributo pode ser acessado com a propriedade **innerHTML**

Exemplo:

<title>Testando DOM</title>

- Elemento nó é <title>
- Texto “Testando DOM” é “filho” do elemento <title>

JS

O método getElementById

A maneira mais comum de acessar um elemento HTML é usar o **id** do elemento.

Exemplo:

```
<!DOCTYPE html>
<html>
	<body>
		<h2>getElementById</h2>
		<p>Retorna o tamanho da string</p>
		<p id="demo"></p>
		<script>
			let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
			document.getElementById("demo").innerHTML = text.length;
		</script>
	</body>
</html>
```

JS

No exemplo anterior, o método **getElementById** usado **id="demo"** para encontrar o elemento.

id="demo“ → Nome de referência para que haja o elo de execução no momento do uso do getElementById e do referencial no HTML.

JS

Exemplo:

```
<html>
        <body>
                <p id="p1">Ola pessoal!</p>
                <script>
                document.getElementById("p1").innerHTML = "Nova
                insercao";
                </script>
        </body>
</html>
```

JS

getElementById ()

O método retorna o elemento que possui o atributo **id** com o valor especificado. Usado para manipular ou obter informações de um elemento em seu documento.

Exemplo:

```
<html>
	<body>
		<h1 id="demo">Passe o mouse sobre mim</h1>
		<script>
			document.getElementById("demo").onmouseover = function() {mouseOver()};
			document.getElementById("demo").onmouseout = function() {mouseOut()};

			function mouseOver() {
			document.getElementById("demo").style.color = "red";
			}
			function mouseOut() {
			document.getElementById("demo").style.color = "black";
			}
		</script>
	</body>
</html>
```

Atividade Prática 5

1. Criar um novo arquivo .html;
2. Criar um parágrafo na tag <body> com a frase “ola pessoal!”;
3. Criar um botão na tag <body> com o dizer “Clique aqui para trocar o texto”;
4. Criar uma função (usar tag <script>) de nome trocacor();
5. O botão ao ser acionado deverá executar a função;

Função:

1. Colocar o parágrafo na cor vermelha;
2. Colocar a nova informação com a frase “Novo texto inserido”;

Atividade Prática 5

Código do exercício:

```
<html>
        <body>
                <p style = "color: aqua" id="p1">Ola pessoal!</p>
                <Button onclick = "trocacor()"> Clique aqui para trocar o texto </Button>
                <script>
                        function trocacor()
                        {
                                document.getElementById("p1").style.color = "red";
                                document.getElementById("p1").innerHTML    =    "texto
                                novo inserido com sucesso";
                        }
                </script>
        </body>
</html>
```

JS

getElementByClasse

O método **getElementsByClass()** retorna uma coleção de todos os elementos no documento com o nome de classe especificado, como um **objeto HTML Collection**.

O objeto **Collection** representa uma coleção de nós.

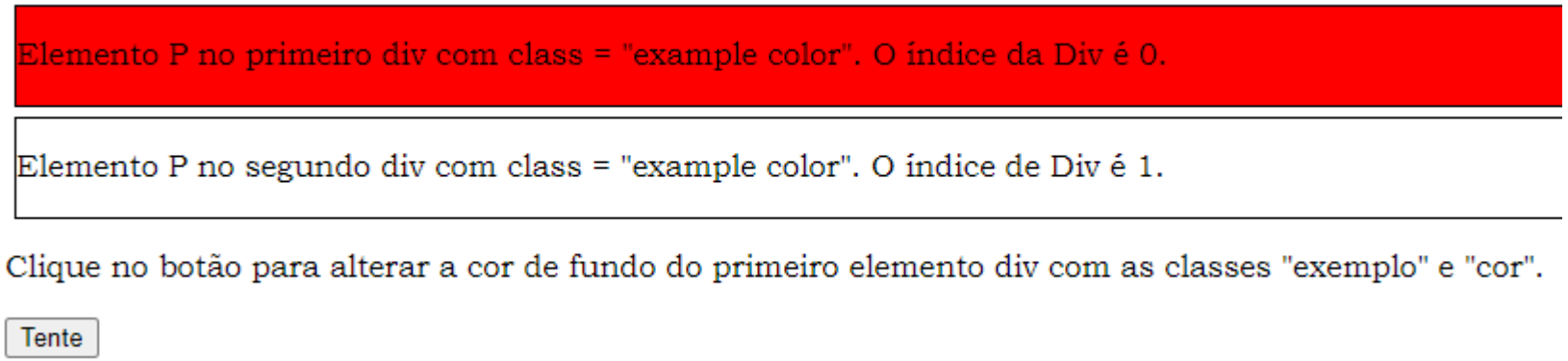
Os nós podem ser acessados por números de índice. O índice começa em 0.

JS

Exemplo:

```
<!DOCTYPE html>
<html>
<body>
   <div class="primeiro teste">Primeiro elemento da classe="primeiro teste".</div>
   <div class="primeiro teste">Segundo elemento da classe="segundo teste".</div>
   <p>Clique no botão para o teste do primeiro elemento da classe="primeiro teste“ (índice
0)</p>
   <button onclick="myFunction()">Clique aqui!</button>
   <script>
      function myFunction() {
      var x = document.getElementsByClassName("primeiro teste");
      x[0].innerHTML = "Alterou a primeira frase";
      }
   </script>
</body>
</html>
```

Atividade Prática 6



1. Criar um arquivo HTML com o nome que quiser;
2. Criar uma tag <style> na tag <head> para configurar as bordas das tags ;
    1. border: 1px solid black;
    2. margin: 5px;
3. Criar duas tags com a referência das classes;
4. Criar uma função para alterar a cor da primeira tag ;
5. Criar uma função para trocar o backgroundcolor da primeira tag ;
6. Criar um botão para executar a função;

Atividade Prática 6

Código do exercício:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      div {
        border: 1px solid black;
        margin: 5px;
        }
    </style>
  </head>
  <body>
    <div class="example color">
    <p>P Elemento P no primeiro div com class = "example color". O índice da Div é 0.</p>
    </div>
    <div class="example color">
    <p>Elemento P no secundario div com class = "example color". O índice da Div é 1.</p>
    </div>
    <p>Clique no botão para alterar a cor de fundo do primeiro elemento div com as classes "exemplo" e "cor".</p>
    <button onclick="myFunction()">Tente</button>
    <script>
      function myFunction() {
      var x = document.getElementsByClassName("example color");
      x[0].style.backgroundColor = "red";
      }
    </script>
  </body>
</html>
```

JS – Material Complementar

Jquery

O **Jquery** é um biblioteca leve, rápida e cheia de recursos para Javascript.

Ele facilita a manipulação de eventos, animações, elementos HTML e utilização de Ajax. Basicamente, ele mudou e facilitou a escrita de códigos em Javascript.

Foi lançado oficialmente em 2006 e possui código aberto. A biblioteca também oferece a possibilidade de criação de plugins sobre ela. Através do jQuery é possível desenvolver aplicações web de alta complexidade.

Site oficial: https://jquery.com

JS

A biblioteca jQuery é um único arquivo JavaScript, e você faz referência a ele com a tag <script> (observe que a tag <script> deve estar dentro da tag <head>):

```
<head>
        <script src="jquery-3.5.1.min.js"></script>
</head>
```

Dica: coloque o arquivo baixado no mesmo diretório das páginas onde deseja usá-lo.

jQuery CDN

Se você não quiser fazer o download e hospedar o jQuery por conta própria, poderá incluí-lo de um CDN (Content Delivery Network).

O Google é um exemplo de alguém que hospeda jQuery:

Google CDN:

```
<head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
```

JS

Sintaxe:

É feita sob medida para **selecionar** elementos HTML e executar alguma **ação** no(s) elemento(s).

A sintaxe básica é: $(seletor). ação()

- Um sinal **$** para definir / acessar jQuery;
- Um **(seletor)** para "consultar (ou localizar)" elementos HTML; e
- Uma **ação jQuery()** a ser realizada no(s) elemento(s).

Exemplos:

$(this).hide() → oculta o elemento atual.
$("p").hide() → oculta todos os elementos <p>.
$(".teste").hide() → oculta todos os elementos com class = "teste".
$("#teste").hide() → oculta o elemento com id = "teste".

JS

O evento de documento pronto

Os métodos jQuery devem ficar dentro de um evento de documento pronto:

```
$(document).ready(function()
{
        // método jQuery ...
});
```

Isso evita que qualquer código jQuery seja executado antes que o documento termine de carregar (esteja pronto). É uma boa prática aguardar que o documento esteja totalmente carregado e pronto antes de trabalhar com ele. Isso também permite que você tenha o código JavaScript antes do corpo do documento, na tag <head>.

JS

O elemento Selector

O seletor de elemento jQuery seleciona elementos com base no nome do elemento.

Você pode selecionar todos os elementos <p> em uma página como esta:

$("p")

Exemplo:

Quando um usuário clica em um botão, todos os elementos <p> ficam ocultos:

```
$(document).ready(function()
{
        $("button").click(function()
        {
                $("p").hide();
        });
});
```

JS

```
<!DOCTYPE html>
<html>
	<head>
		<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
		<script>
			$(document).ready(function(){
				$("button").click(function(){
					$("p").hide();
				 });
			});
		</script>
	</head>
	<body>
		<h2>Titulo da pagina</h2>
		<p>Primeiro paragrafo</p>
		<p>Segundo paragrafo</p>

		<button>Clique aqui para esconder os paragrafos</button>
	</body>
</html>
```

JS

O seletor #id

O seletor jQuery usa o **atributo id** de uma tag HTML para encontrar o elemento específico.
Um **id** deve ser único em uma página, então você deve usar o seletor **#id** quando quiser encontrar um elemento único e único.
Para encontrar um elemento com um **id** específico, escreva um caractere **hash**, seguido pelo **id** do elemento HTML:

```
$("#teste")          a palavra teste é o id, então temos #teste
```

Exemplo:

Quando um usuário clica em um botão, o elemento **com id = "teste"** fica oculto:

```
$(document).ready(function()
        {
        $("button").click(function()
        {
                $("#teste").hide();
        });
});
```

JS

```
<!DOCTYPE html>
<html>
	<head>
		<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
		<script>
			$(document).ready(function(){
				$("button").click(function(){
					$("#teste").hide();
				});
			});
		</script>
	</head>
	<body>
		<h2>Titulo da pagina</h2>
		<p>Isto e um paragrafo</p>
		<p id="teste">Este e outro paragrafo.</p>

		<button>Clique aqui!</button>
	</body>
</html>
```

JS

O seletor .class

O **.classseletor** jQuery encontra elementos com uma classe específica.

Para encontrar elementos com uma classe específica, escreva um caractere de ponto, seguido do nome da classe:

```
$(".teste")
```

Exemplo:

Quando um usuário clica em um botão, os elementos com **class = "teste"** serão ocultados:

```
$(document).ready(function(){
        $("button").click(function(){
                $(".teste").hide();
        });
});
```

JS

```
<!DOCTYPE html>
<html>
	<head>
		<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
		<script>
			$(document).ready(function(){
				$("button").click(function(){
					$(".teste").hide();
				});
			});
		</script>
	</head>
	<body>
		<h2 class="teste">Titulo do paragrafo</h2>
		<p class="teste">Primeiro paragrafo</p>
		<p>Segundo paragrafo</p>

		<button>Clique aqui!</button>
	</body>
</html>
```

JS

Sintaxe	Descrição
$("*")	Seleciona todos os elementos
$(this)	Seleciona o elemento html atual
$("p.intro")	Seleciona todos os elementos <p> com a classe="intro"
$("p:first")	Seleciona o primeiro elemento <p>
$("ul li:first")	Seleciona o primeiro elemento do primeiro
$("ul li:first-child")	Seleciona o primeiro elemento de todos os
$("[href]")	Seleciona todos os elemnetos com o atributo href
$("a[target='_blank']")	Seleciona todos os elementos <a> com o alvo de valor igual a "_blank"
$("a[target!='_blank']")	Seleciona todos os elementos <a> com o alvo de valor diferente de "_blank"
$(":button")	Seleciona todos os elementos <button> e elementos <input> elementos do tipo type="button"
$("tr:even")	Seleciona todos os elementos <tr>
$("tr:odd")	Seleciona todos os elementos de linha ímpar <tr>

JS

Funções em um arquivo separado

Se o seu site contém muitas páginas, e você deseja que suas funções jQuery sejam fáceis de manter, você pode colocar suas funções jQuery em um arquivo **.js separado.**

Quando demonstramos o jQuery neste tutorial, as funções são adicionadas diretamente na **<head>** seção. No entanto, às vezes é preferível colocá-los em um arquivo separado, como este (use o atributo src para se referir ao arquivo .js):

Exemplo:

```
<head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery
        .min.js"></script>
        <script src="my_jquery_functions.js"></script>
</head>
```

JS

Métodos de evento jQuery

jQuery é feito sob medida para responder a eventos em uma página HTML.

O que são eventos?

Todas as diferentes ações dos visitantes às quais uma página da web pode responder são chamadas de eventos.

Um evento representa o momento preciso em que algo acontece.

Exemplos:

- mover o mouse sobre um elemento;
- selecionando um botão de rádio; e
- clicando em um elemento.

JS

Aqui estão alguns eventos DOM comuns:

Eventos do mouse	Eventos do teclado	Eventos do formulário	Eventos do Document/Window
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

JS

Sintaxe jQuery para métodos de evento

Em jQuery, a maioria dos eventos DOM tem um método jQuery equivalente.

Para atribuir um evento de clique a todos os parágrafos de uma página, você pode fazer o seguinte:

$("p").click();

A próxima etapa é definir o que deve acontecer quando o evento for disparado. Você deve passar uma função para o evento:

```
$("p").click(function(){
        // ação executada aqui!
});
```

JS

Métodos de evento jQuery comumente usados

$ (document) .ready ()

O método $(document).ready() → nos permite executar uma função quando o documento está totalmente carregado.

click()

O método click() → anexa uma função de manipulador de eventos a um elemento HTML.

A função é executada quando o usuário clica no elemento HTML.

JS

```
<!DOCTYPE html>
<html>
	<head>
		<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
		<script>
			$(document).ready(function(){
				$("p").click(function(){
					$(this).hide();
				});
			});
		</script>
		</head>
		<body>
			<p>Se clicar em mim eu desapareco</p>
			<p>Clique em mim</p>
			<p>Clique em mim também, por favor</p>
		</body>
</html>
```

JS

Definir conteúdo - text (), html () e val ()

Métodos para **definir o conteúdo** :

- text() → Define ou retorna o conteúdo de texto dos elementos selecionados;
- html() → Define ou retorna o conteúdo dos elementos selecionados (inclui marcação HTML); e
- val() → Define ou retorna o valor dos campos do formulário;

Exemplo:

```
$("#btn1").click(function(){
        $("#test1").text("Maravilha de teste!");
});

$("#btn2").click(function(){
        $("#test2").html("<b>Seja bem-vindo!</b>");
});

$("#btn3").click(function(){
        $("#test3").val("vamos jogar");
});
```

JS

```
<!DOCTYPE html>
<html>
	<head>
		<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
		<script>
			$(document).ready(function(){
				$("#btn1").click(function(){
					alert("Texto: " + $("#teste").text());
				});
				$("#btn2").click(function(){
					alert("codigo HTML: " + $("#teste").html());
				});
			});
		</script>
	</head>
	<body>
		<p id="teste">Isto esta <b>em negrito</b> em algum texto do paragrafo</p>
		<button id="btn1">Exibir o texto</button>
		<button id="btn2">exibir o html</button>
	</body>
</html>
```