JS 5

Para criar novos elementos na árvore DOM você deve antes criar o nodo elemento e depois pendurá-lo em um elemento já existente. Veja o exemplo

```
<body> <div id="div1">
<p id="p1">Um parágrafo</p>
<p id="p2">Outro parágrafo</p></div>
<script> var para=document.createElement("p");
var node=document.createTextNode("O novo");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para);
</script> </body>
```

No exemplo acima, para criar um elemento `<p>` novo o comando é: var para = document.createElement("p"); Para adicionar texto ao elemento `<p>` deve-se fazer: var node = document.createTextNode("O novo"); Para pendurar o texto no elemento `<p>`: para.appendChild(node); Para pendurar o novo elemento em um já existente, primeiro deve-se achar um elemento já existente: var element = document.getElementById("div1"); e depois pendurar o novo: element.appendChild(para).

O método `appendChild()` no exemplo anterior, pendura o novo elemento como o último filho do pai. Se você quer fazer diferente pode usar o método `insertBefore()`. Veja no exemplo

```
<body> <div id="div1">
<p id="p1">Um parágrafo</p>
<p id="p2">Outro parágrafo</p>
</div> <script>
var para=document.createElement("p");
var node=document.createTextNode("O novo");
para.appendChild(node);
var element=document.getElementById("div1");
var child=document.getElementById("p1");
element.insertBefore(para,child);
</script> </body>
```

Para remover elementos HTML já existentes, primeiro deve-se achar o pai do elemento. Veja

```
<body> <div id="div1">
<p id="p1">Um parágrafo</p>
<p id="p2">Outro parágrafo</p>
</div> <script>
var parent=document.getElementById("div1");
var child=document.getElementById("p1");
parent.removeChild(child);
</script> </body>
```

O exemplo explicado: Este HTML possui um elemento `div>` com dois filhos (ambos elementos `<p>`). Para achar o elemento cuja `id` é `p1` o comando é: var parent = document.getElementById("div1"); Daí, para achar o elemento `<p>` cuja `id` é `p1`: var child = document.getElementById("p1"); e para remover o filho do pai: parent.removeChild(child);

Note que seria bem bom poder remover o filho sem conhecer o pai, mas o DOM não aceita. Para remover um filho é imprescindível conhecer seu pai. Mas espere: sempre tem um jeito. Pode-se usar a propriedade `parentNode` para achar o pai e fica:

```
var child = document.getElementById("p1");
child.parentNode.removeChild(child);
```

Para substituir um elemento HTML usa-se o método `replaceChild()` Veja como

```
<body> <div id="div1">
<p id="p1">Um parágrafo</p>
<p id="p2">Outro parágrafo</p>
</div> <script>
var parent=document.getElementById("div1");
var child=document.getElementById("p1");
var para=document.createElement("p");
var node=document.createTextNode("O novo");
para.appendChild(node);
parent.replaceChild(para,child);
</script> </body>
```

O método `getElementsByTagName()` retorna uma lista de nodos. É uma coleção na forma de um array. Por exemplo, o código abaixo retorna todos os nodos `<p>` em um documento

```
var x=document.getElementsByTagName("p");
```

Os nodos podem ser acessados pelo seu índice. Para acessar o segundo nodo `<p>`, deve-se fazer

```
y = x[1];
```

Veja um exemplo completo:

```
<body> <p>Oi mundo</p>
<p>O DOM é muito útil</p>
<p id="oba"></p>  <script>
var myNodelist=document
  .getElementsByTagName("p");
document.getElementById("oba").innerHTML=
"O innerHTML do 2. parágrafo é: " +
myNodelist[1].innerHTML;
</script> </body>
```

Para trocar a cor de fundo de todos os elementos `<p>` da lista de nodos:

```
<body> <p>Um elemento p</p>
<p>Outro elemento p</p>
<p>Ainda outro elemento p - Clique no botão
para trocar a cor de fundo de todos os p
deste documento </p>
<button onclick="fun1()">Tente</button>
<script>
function fun1() {
 var ld = document.getElementsByTagName("p");
 var i;
 for (i = 0; i < ld.length; i++) {
  ld[i].style.backgroundColor = "red"; } }
</script> </body>
```

Uma observação é que a lista de nodos parece um array, tem cheiro e aspecto de array mas não é verdadeiramente um. Isto significa que não dá para usar métodos de array (como `valueOf()` ou `join()`) nela.

Existe um parente do DOM que é o BOM (browser object model). Este modelo permite ao JS "conversar" com o browser. Infelizmente o BOM não é padronizado, mas os browsers modernos têm implementado boa parte de sua funcionalidade. Por exemplo, o objeto `window` é suportado por todos os browsers. Ele representa a janela do browser. Todos os objetos JS, as funções JS e as variáveis JS são membros do objeto `window`. As variáveis globais são propriedades do objeto `window`. As funções globais são métodos do objeto `window`. Mesmo o objeto documento (do DOM do HTML) é uma propriedade do objeto `window`. Logo os dois comandos abaixo são equivalentes

```
window.document.getElementById("header");
document.getElementById("header");
```

Para todos os browsers (IE só a partir do 9) as propriedades para conhecer o tamanho da janela são

```
<body> <p id="oba"></p>
<script>
var w = window.innerWidth
var h = window.innerHeight
var x = document.getElementById("oba");
x.innerHTML = "Largura: " + w +
", Altura: " + h + ".";
</script> </body>
```

Para crossbrowser, incluindo versões antigas do IE veja `www.w3schools.com/js/js_window.asp`

Para criar um botão de *Back* que significa voltar à última página visitada, o código é:

```
<html> <head> <script>
function goBack() {
    window.history.back() }
</script> </head> <body>
<input type="button" value="Back"
   onclick="goBack()">
</body> </html>
```

Já para avançar à próxima página (da lista de histórico do browser) o código é:

```
<html> <head> <script>
function goForward() {
    window.history.forward() }
</script> </head> <body>
<input type="button" value="Forward"
   onclick="goForward()">
</body> </html>
```

Para obter uma informação a partir de uma *prompt box* o código é

```
<body><p>Aperte o botão</p>
<button onclick="fun1()">Tente</button>
<p id="oba"></p>
<script>
function fun1() {
 var pessoa = prompt("Seu nome", "Zé Mané");
 if (pessoa != null) {
  document.getElementById("oba").innerHTML =
  "Oi " + pessoa + " Como vai?"; } }
</script> </body>
```

No JS pode-se executar algum código a intervalos específicos de tempo. Há dois métodos aqui: `setInterval()` usado para executar uma função a intervalos específicos para sempre e `setTimeout()` que executa uma função uma única vez após esperar um certo tempo. (ambos são métodos do objeto window do DOM HTML). Veja um exemplo

```
<body>  <p>Aperte "Tente". Espere 3 segundos.
Sua página vai alertar "Oi"</p>
<p>Quando vc fechar a caixa de alerta,
uma nova caixa vai
aparecer a cada 3 segundos</p>
<p>Para sempre, até fechar a janela</p>
<button onclick="setInterval(function()
{alert('Oi')},3000);">Tente</button>
</body>
```

Para interromper a execução, a síntaxe é

```
window.clearInterval(intervalVariable)
```

No exemplo abaixo a execução ocorre uma única vez

```
<body> <p>Clique "Tente". Espere 2 segundos.
A página vai escrever "Oi"</p>
<button onclick="setTimeout(function()
{alert('Oi')},2000);">
Tente</button> </body>
```

Para parar o tempo neste segundo caso a síntaxe é

```
window.clearTimeout(timeoutVariable)
```

Exercício 1

Escreva arquivos HTML com todos os exemplos desta folha.

☞ Para você fazer

1. Escreva aqui e se não couber continue no verso desta folha um resumo das coisas estudadas nesta aula:

   _____

   _____

   _____

   _____

   _____

   _____

   _____

2. Imprima os arquivos exemplos desta folha grampeie-os nesta folha e devolva tudo ao professor.

.