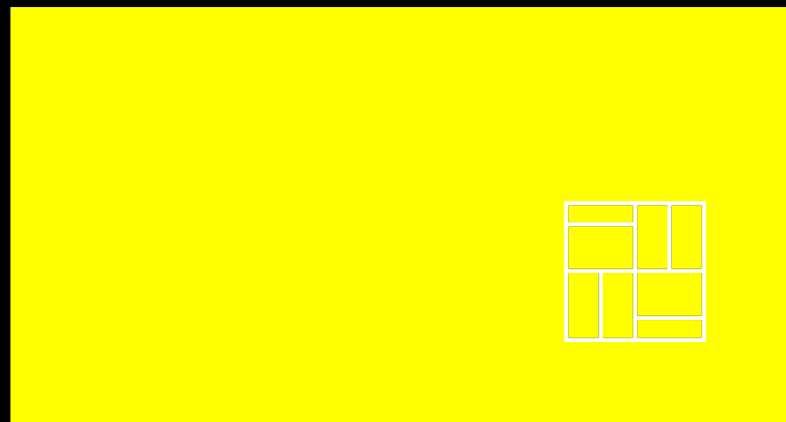
DOM

O DOM (Document Object Model) é a representação de dados dos objetos que compõem a estrutura e o conteúdo de um documento na Web.

Em Javascript, o DOM é usado para aceder ao documento e aos seus elementos.



Estrutura

A DOM representa um documento com uma árvore lógica. Cada ramo da árvore termina em um nó, e cada nó contém objetos.

WINDOW

A window não faz parte da DOM, mas é o interface que contém a DOM e a sua estrutura.

DOCUMENT

Para cada página carregada no browser, existe um objeto Document.
A interface Document é a raíz do conteúdo da Página.

NODE

Cada objecto que existe na estrutura DOM é um node.
Textos, elementos, atributos e inclusive os comentários são considerados nodes.

Estrutura

ELEMENT

Um objecto element representa um elemento HTML, como um *<p>*, * *, *<a>*, entre muitos outros

NODELIST

Um nodelist é um conjunto de elements que pertencem a um document.
Cada um dos elementos da nodelist pode ser acedida por um índice, tal como se faz num array.

```
myNodeList[1]
```

ATTRIBUTES

In the DOM, an attribute object represents an HTML attribute. An attribute always belongs to an HTML element.

```
var aName = element.attributes[0].name;
```

Selectores

Os seletores vão permitir escolher um ou mais nós da DOM para que se possa interagir/manipular/extrair conteúdo.

GETELEMENTBYID(id)

Procura o primeiro node cujo atributo id seja exatamente igual à string enviada. Retorna null caso não encontre nenhum elemento.

document.getElementById('teste');

GETELEMENTSBYCLASSNAME(classname)

Procura os nodes cujo atributo class contém a string enviada. Retorna [] (array vazio) caso não encontre nenhum elemento.

document.getElementsByClassName(className);

GETELEMENTSBYTAGNAME(tagName)

Procura os nodes cujo o nome da sua tag seja exatamente igual à
string enviada. Retorna [] caso não encontre nenhum elemento.

document.getElementsByTagName(tagName);

Selectores

QUERYSELECTOR(selectores)

Retorna o primeiro element dentro do documento que corresponde ao seletor, ou grupo de seletores (CSS-selectores) especificados. Se não há elementos que correspondem, null é devolvido como resultado.

```
<div id="teste"></div>

document.querySelector("#teste");
```

QUERYSELECTORALL(selectores)

Faz o mesmo que o querySelector mas devolve todos os elementos que correspondam à procura.
Retorna [] (array vazio) caso não encontre nenhum elemento.

```
<div class="teste"></div>
<div class="teste bold"></div>

document.querySelectorAll(“.teste”);
```

Exercício

DOM

- Copia o seguinte código html para o um ficheiro:

```
<button type="button">Elementos</button>
<p id="teste">Teste</p>
<button type="button">Primeiro</button>

<div class="container">
        <h1><time>00:00:00</time></h1>
        <button id="start">Start</button>
        <button id="stop">Stop</button>
        <button id="reset">Reset</button>
</div>
```

- Faz *console.log* de todos os button.

- Faz *console.log* dos button que fazem parte da div com a class container.

- Faz *console.log* do p com o id teste.

Criar e adicionar elementos

O Javascript disponibiliza diversas funções que vão permitir criar e adicionar elementos e nós à nossa estrutura HTML. Consideremos o exemplo abaixo que cria vários elementos:

```
function adicionarElemento() {
    var novaDiv = document.createElement("div");
    var conteudo = document.createTextNode("Esta frase está a ser criada dinamicamente");
    novaDiv.appendChild(conteudo);
    document.body.appendChild(novaDiv);
}
```

Criar elementos

CREATEELEMENT(tagName)

Cria o elemento HTML especificado.

```
var elemento = document.createElement('div');
```

CREATETEXTNODE(conteudo)

Cria um nó de tipo texto (string).

```
document.createTextNode("Exemplo");
```

CREATEATTRIBUTE(atributo)

Cria e retorna um novo *atributo* a ser aplicado a algum elemento.

```
var a = document.createAttribute("atributo_novo");
a.value = "valor do atributo novo";
```

Adicionar elementos

APPENDCHILD(elemento)

Adiciona um node ao fim da lista de nodes do parente especificado.

```
var newDiv = document.createElement("div");
document.body.appendChild(newDiv);
```

APPEND(conteudo)

Adiciona qualquer elemento ao fim da lista de nodes do parente especificado. Os elementos a serem adicionados podem ser objectos, strings ou nodes, ao contrário do método appendChild que apenas aceita nodes.

```
var newDiv = document.createElement("div");
newDiv.append(“uma string”);
```

PREPEND(conteudo)

Adiciona qualquer elemento ao início da lista de nodes do parente especificado.

```
var newDiv = document.createElement("div");
newDiv.prepend(“uma string”);
```

Adicionar elementos

SETATTRIBUTENODE(atributo)

Este método adiciona um nó de tipo atributo a um elemento.

```
var div1 = document.createElement('div');
var atributo = document.createAttribute('atributo');
atributo.value = 'teste';
div1.setAttributeNode(atributo);
```

SETATTRIBUTE(nome, valor)

Este método determina um atributo e o seu valor a um elemento. Se o atributo já existir, o seu valor é actualizado. Se não existir, é adicionado.

```
var div1 = document.createElement('div');
div1.setAttribute('id', 'teste');
```

Exercício

DOM

- Cria um ficheiro html;
- Cria uma tag script para o código js;
- Cria uma *div* e adiciona ao body.
- Adiciona à div anteriormente criada o seguinte texto: *Isto é apenas para um exercício*;
- Adiciona ao fim da div anteriormente criada o seguinte texto: *muito difícil*;

EVENTOS DOM

Eventos são ações ou ocorrências que acontecem no sistema que estamos desenvolvendo, no qual este alerta sobre essas ações para que se possa responder de alguma forma, se desejado.

Os eventos podem representar tudo, desde interações básicas do usuário até notificações automatizadas de coisas que acontecem no modelo de renderização.

Eventos

Em Javascript existem diversos tipos de eventos definidos:

ONBLUR

O evento *blur* dispara quando um elemento perde o *focus* (foco).

ONFOCUS

O evento *focus* dispara quando um elemento ganha o *focus* (foco).

ONLOAD

O evento *load* dispara quando um recurso ou elemento termina o seu carregamento. Por recursos referimo-nos a ficheiros: imagens, css, documentos, etc.

ONCLICK

O evento *click* dispara quando um elemento é clicado pelo utilizador.

ONKEYPRESS

O evento *keyPress* dispara quando um utilizador pressiona continuamente uma tecla sobre o elemento.

ONKEYDOWN

O evento *keyDown* dispara quando um utilizador pressiona uma tecla sobre o elemento.

Eventos

ONKEYUP

O evento *keyUp* dispara quando um utilizador liberta o pressionar de uma tecla sobre o elemento.

ONMOUSEOVER

O evento *mouseOver* dispara quando o utilizador passa o rato (ou outro dispositivo semelhante) sobre o elemento.

ONMOUSEOUT

O evento *mouseOut* dispara quando o utilizador retira o rato (ou outro dispositivo semelhante) sobre o elemento.

ONMOUSEDOWN

O evento *mouseDown* dispara quando o utilizador pressiona num botão do rato (ou outro dispositivo semelhante) sobre o elemento.

ONMOUSEUP

O evento *mouseUp* dispara quando o utilizador retira o pressionar de um botão do rato (ou outro dispositivo semelhante) sobre o elemento.

Eventos

ONSUBMIT

O evento *submit* dispara imediatamente antes de um formulário ser submetido.

ONRESET

O evento *reset* dispara imediatamente antes de um formulário ser limpo (reset).

ONCHANGE

O evento *change* dispara quando o valor de um elemento muda. Está relacionado sobretudo com elementos relativos a formulários, como *inputs*, *selects*, *radio buttons*, etc.

ONRESIZE

O evento *resize* dispara quando o utilizador redimensiona a janela do browser (*window*).

ONSCROLL

O evento *scroll* dispara quando o utilizador faz scroll no *document* ou nalgum elemento.

Eventos - como aplicar

Existem várias maneiras de se aplicar eventos aos elementos HTML:

INLINE

A linguagem Javascript permite que o código a ser executado num evento seja informado diretamente na propriedade que lhe dá acesso.

Essa forma é utilizada quando se tem poucas instruções a serem executadas e quando as expressões são curtas e de fácil compreensão.

```
<button id="btn" onclick=”console.log(‘teste’)”>
    Clique aqui
</button>

<button onclick=”this.setAttribute(‘class’, ‘clicked’)”>
    Clique aqui
</button>
```

EVENT HANDLER

Os event handlers são funções que contém o código a ser executado na ocorrência de um evento. Em Javascript, podemos criar uma função utilizando a sintaxe padrão e fazer a chamada a essa função na propriedade de evento, informando seu nome e possíveis parâmetros no lugar onde se colocaria o código diretamente.

```
<script>
    function mostrarMensagem() {
        console.log(‘teste’);
    }
</script>
<button onclick="mostrarMensagem();">
    Clique aqui
</button>
```

Exercício

EVENTOS

- Cria um botão com evento *onclick*;
- Cria um evento *onload* na tag body;
- Cria um formulário com:
  - 1 campo de texto com um evento *onchange*;
  - um botão de submit;
  - um botão de reset;
  - adiciona o event *onsubmit* ao formulário;
  - adiciona o event *onreset* ao formulário;
- Cria um botão com o evento *mouseover* e *mouseout*; Estes eventos devem chamar cada um uma função (um event handler);
- Cria um input de tipo texto e adiciona os eventos *onfocus* e *onblur*. Estes eventos devem chamar cada um uma função (um event handler).

Eventos - como aplicar

Existem outras maneira sde se aplicar eventos aos elementos no script de javascript:

EVENTOS DOS ELEMENTS

Cada *element* possui formas de lidar com os eventos que lhe pertencem:

```
var form = document.getElementById('form');
form.onsubmit = function () {
        console.log('submetido');
};
```

EVENT LISTENERS

Event Listener é um manipulador de eventos que tem a capacidade de adicionar ou remover um evento sobre qualquer elemento.

Eventos - Event Listeners

O Event Listener disponibiliza duas funções principais, são elas:

ADDEVENTLISTENER(evento, funcao)

Adiciona uma função que será disparada quando ocorrer determinado evento no objeto.
Podem ser adicionar múltiplos event listeners a um elemento.

```
var div1 = document.createElement('div');
div1.addEventListener('click', funcaoParaEvento);
```

REMOVEEVENTLISTENER(evento, funcao)

Remove um listener previamente adicionado a um elemento e retorna true em caso de sucesso.

```
div1.removeEventListener('click', funcaoParaEvento);
```

Eventos - Prevent default

EVENT.PREVENTDEFAULT()

Previne que a ação por defeito desse evento seja executada.
Exemplo: Um formulário ao ser submetido refresca a página, isso pode ser evitado usando este método.

```
div1.addEventListener('click', funcao);

function funcao (event) {
    event.preventDefault();
}


div1.onclick = function (event) {
    event.preventDefault();
}
```

Eventos - Pseudo eventos

SETINTERVAL(funcao, tempoMilissegundos*)*

Método que chama uma função (*funcao*) em intervalos (*tempoMilissegundos*) especificados em milissegundos.

```
setInterval(function () {
    console.log('isto demorou 5 segundos')
}, 5000);
```

CLEARINTERVAL(interval)

O método *setInterval* continua chamando a função até a janela ser fechada ou até clearInterval ser chamado.

```
var interval = setInterval(function () {
    console.log('isto demorou 5 segundos')
}, 5000);

clearInterval(interval);
```

SETTIMEOUT(funcao, tempoMilissegundos)

O setInterval()método continua chamando a função até clearInterval()ser chamado ou a janela ser fechada.

```
setTimeout(function () {
    console.log('isto demorou 5 segundos')
}, 5000);
```

Exercício

EVENTOS PSEUDO EVENTOS

- Cria um formulário com:
  - um botão de submit;
  - adiciona o event *onsubmit* ao formulário que previna o seu comportamento por defeito e faça um console.log e uma mensagem à escolha.

- Copia o seguinte html:

  <button>Button 1</button>
  <button>Button 2</button>

  Adiciona a ambos os botões event listeners para *mouseover* e *mouseout*;

- Adiciona um *setTimeout* que adicione um id ao primeiro botão adicionado no segundo passo do exercício;