PADO

Labs

INTRODUÇÃO JAVASCRIPT

Desenvolvimento Web - Aula VI

Contato

Professor


David Krepsky


david@hausenn.com.br


DKrepsky

Monitor


Kevin Araujo


kevin.araujo@hausenn.tech


NivekDesign

- Introdução JavaScript

- Caracterśticas da Linguagem
- Console do Navegador

<Caracterśticas da Linguagem>

O JavaScript, como o próprio nome sugere, é uma linguagem de scripting. Uma linguagem de scripting é comumente definida como uma linguagem de programação que permite ao programador controlar uma ou mais aplicações de terceiros. No caso do JavaScript, podemos controlar alguns comportamentos dos navegadores através de trechos de código que são enviados na página HTML.

<Caracterśticas da Linguagem>

Uma característica comum nas linguagens de scripting é que normalmente elas são linguagens interpretadas, ou seja, não dependem de compilação para serem executadas. Essa característica é presente no JavaScript: o código é interpretado e executado conforme é lido pelo navegador, linha a linha, assim como o HTML

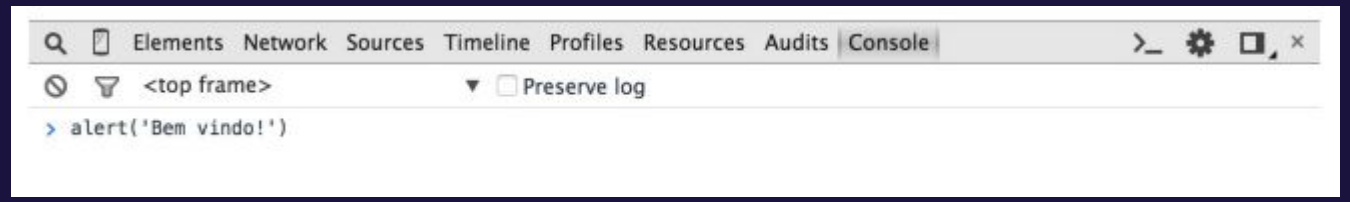
O JavaScript possui grande tolerância a erros.

<Console do Navegador>

Existem várias formas de executar códigos JavaScript em uma página. Uma delas é executar códigos no que chamamos de **Console.** A Maioria dos navegadores desktop já vem com essa ferramenta instalada.

<Developer Tools>

O console faz parte de uma série de ferramentas embutidas nos navegadores especificamente para nós que estamos desenvolvendo um site. Essa série de ferramentas é que chamamos de Developer Tools.



<Sintaxe Básica >

Variáveis: para armazenarmos um valor para uso posterior, podemos criar uma variável

> **var** *resultado* = 102 / 17;
undefined
No exemplo acima, guardamos o resultado de 102 / 17 na variável *resultado*. O resultado de criar uma variável é sempre **undefined.** Para obter o valor que guardamos nela ou mudar o seu valor, podemos fazer o seguinte:
> resultado
6

> resultado = resultado + 10
16

< resultado
16

<Sintaxe Básica >

Também podemos alterar o valor de uma variável usando as operações básicas com uma sintaxe bem compacta:

```
> var idade = 10;
> idade += 10;          // idade vale 20
> idade -= 5;           // idade vale 15
> idade /= 3;           // idade vale 5
> idade *= 10;          // idade vale 50
```

<Number>

Com esse tipo de dados é possível executar todas as operações que vimos anteriormente:

```
var pi = 3.14159;
var raio = 20;
var perimetro = 2 * raio;
```

<String>

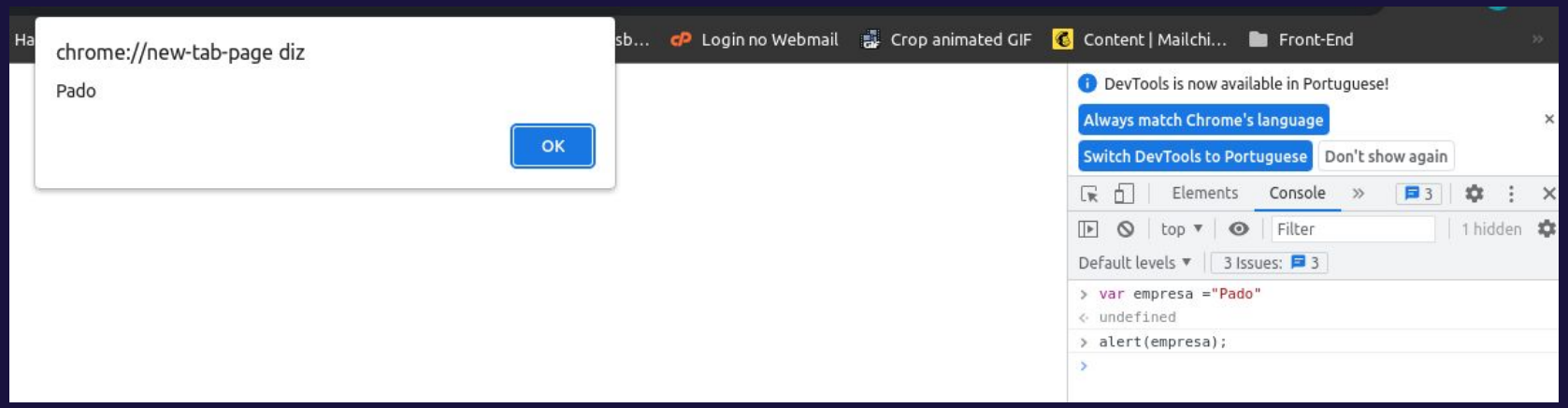
Não são apenas números que podemos salvar numa variável. O JavaScript tem vários tipos de dados.

Uma string em JavaScript é utilizada para armazenar trechos de texto:

var empresa = “Pado”;

Para exibirmos o valor da variável empresa fora do console, podemos executar o seguinte comando:

alert(empresa)



<Automatic semicolon insertion (ASI)>

É possível omitir o ponto e vírgula no final de cada declaração. A omissão de ponto e vírgula funciona no JavaScript devido ao mecanismo *automatic semicolon insertion (ASI).*

<A tag Script>

Para inserirmos um código JavaScript em uma página, é necessário utilizar a tag **<script>** :

```
<script>
alert (“Olá, Mundo!”);
</script>
```

A Tag <script> pode ser declarada dentro da tag <head> assim como na tag <body>, mas devemos ficar atentos, porque o código é lido imediatamente dentro do navegador.

<A tag Script>

Ao ser executado, o script trava o processamento da página. Um script que demore um pouco mais para ser executado ou que exija alguma interação do usuário como uma confirmação, seria interessante carregar a página toda primeiro antes de sua execução por uma questão de performance.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Aula de JS</title>
    <script>
       alert("Olá, Mundo!");
    </script>
  </head>
  <body>
    <h1>JavaScript</h1>
    <h2>Linguagem de programação</h2>
  </body>
</html>
```

<input type="number">

Para fazer isso, basta removermos o script do <head> , colocando-o no final do <body> :

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Aula de JS</title>
  </head>
  <body>
    <h1>JavaScript</h1>
    <h2>Linguagem de Programação</h2>
    <script>
      alert("Olá, Mundo!");
    </script>
  </body>
</html>
```

Devemos sempre colocar o script antes de fecharmos a tag <body>

<JavaScript Arquivo Externo>

Com a separação do script em arquivo externo é possível reaproveitar alguma funcionalidade em mais de uma página.

```
<script type="text/javascript" src="js/hello.js"></script>

alert("Olá, Mundo!");
```

<Mensagens no console>

Quando queremos dar uma olhada no valor de alguma variável ou resultado de alguma operação durante a execução do código. Nesses casos, poderíamos usar o alert. Porém, se esse conteúdo deveria somente ser mostrado para o desenvolvedor, o console do navegador pode ser utilizado no lugar do alert para imprimir essa mensagem:

```
var mensagem = "Olá mundo";
console.log(mensagem);
```

<DOM>

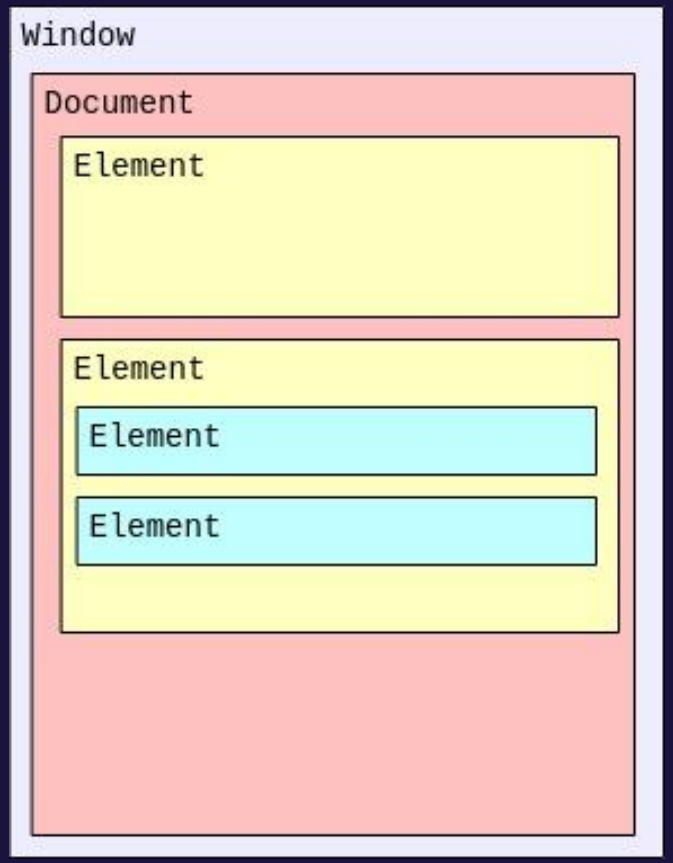
DOM: Sua página no mundo JavaScript

Para permitir alterações na página, ao carregar o HTML da página, os navegadores carregam em memória uma estrutura de dados que representa cada uma das nossas tags no JavaScript. Essa estrutura é chamada de DOM (Document Object Model).

Essa estrutura pode ser acessada através da variável global *document*.

<Element HTML>

HTMLElement interface representa qualquer elemento HTML . Alguns elementos implementam diretamente essa interface, enquanto outros a implementam por meio de uma interface que a herda.



https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement

<querySelector>

Em primeiro lugar precisamos acessar no JavaScript o elemento que queremos alterar:

```
document.querySelector("h1")
```

Esse comando usa os seletores CSS para encontrar os elementos na página.

Usamos um seletor de nome de tag mas poderíamos usar outros como:

```
document.querySelector("#id")
```

```
document.querySelector(".class")
```

<Elemento da página como variável>

Se for utilizar várias vezes um mesmo elemento da página, é possível salvar o resultado de qualquer querySelector numa variável:

```
var titulo = document.querySelector("h1")
```

Executando no console, você vai perceber que o elemento correspondente é selecionado. Podemos então manipular seu conteúdo:

```
titulo.textContent
```

Essa propriedade, pode receber valores e ser alterada:

```
titulo.textContent = "Novo título"
```

<querySelectorAll>

Selecionar vários elementos na página. Várias tags com a classe **.padolabs** por exemplo. Se o retorno esperado é mais de um elemento, usamos o *querySelectorAll* que devolve uma lista de elementos (array).

document.querySelectorAll(“.padolabs”)

podemos então acessar elementos nesta lista através da posição dele que começa em 0 e usando o colchetes:

document.querySelectoAll(“.padolabs”)[0]

<Function>

Para guardarmos um código para ser executado em algum outro momento, por exemplo, quando o usuário clicar num botão, é necessário utilizar alguns recursos do JavaScript no navegador. Primeiro vamos criar uma *função(function):*

```
function mostraAlerta() {
  alert("Funciona!");
}
```

Ao criar uma função, guardamos o que estiver dentro da função, e esse código guardado só será executado quando chamarmos a função:

```
function mostraAlerta() {
  alert("Funciona!");
}

// fazendo uma chamada para a função mostraAlerta, que será executada nesse momento
mostraAlerta()
```

Para chamar a função mostraAlerta só precisamos utilizar o nome da função e logo depois abrir e fechar parênteses.

<Function>

Para que nossa função seja chamada quando o usuário clicar no botão da nossa página, precisamos do seguinte código:

```
function mostraAlerta() {
  alert("Funciona!");
}

// obtendo um elemento através de um seletor de ID
var botao = document.querySelector("#botaoEnviar");

botao.onclick = mostraAlerta;
```

Primeiramente foi necessário selecionar o botão e depois definir no *onclick* que o que vai ser executado é a função *mostraAlerta. Será sempre essa mesma forma para qualquer código que tenha que ser executado após alguma ação do usuário em algum elemento. O que mudará sempre é qual elemento você está selecionando, a qual evento você está reagindo e qual função será executada.*

<Eventos existentes>

onInput: Quando um elemento input tem seu valor modificado.
onclick: Quando ocorre um click com o mouse.
ondblclick: Quando ocorre dois clicks com o mouse.
onmousemove: Quando mexe o mouse.
onmousedown: Quando aperta o botão do mouse.
onmouseup: Quando solta o botão do mouse.
onkeypress: Quando pressionar uma tecla.
onkeyUp: Quando soltar uma tecla.
onblur: Quando um elemento perde foco.
onfocus: Quando um elemento ganha foco.
onchange: Quando um input select ou textarea tem seu valor alterado.
onload: Quando a página é carregada.
onunLoad: Quando a página é fechada.
onsubmit: Dispara antes de submeter o formulário (útil para realizar validações).

setTimeout()

O método global setTimeout()define um cronômetro que executa uma função ou trecho de código especificado

O setTimeout() método chama uma função após alguns milissegundos. 1 segundo = 1000 milissegundos.

O setTimeout() é executado apenas uma vez. Se você precisar de execuções repetidas, use setInterval()em vez disso.

Teste setTimeout()


```html
<!DOCTYPE html>
<html>
<body>


<h2>metodo setTimeout() </h2>

<p>clique no botão, Espere 3 segundos para mostrar um alerta na tela.</p>

<button onclick="myFunction()">Try it</button>

<script>
let timeout;

function myFunction() {
  timeout = setTimeout(alertFunc, 3000);
}

function alertFunc() {
  alert("PADOLABS");
}
</script>

</body>
</html>
```

setInteval()

O setInterval() método chama uma função em intervalos especificados (em milissegundos 1 segundo = 1000 milissegundos).

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

teste setInterval()


```html
<!DOCTYPE html>
<html>
<body>


<p id="horas"></p>

<button onclick="myStopFunction()">Parar horas</button>

<script>
const myInterval = setInterval(myTimer, 1000);

function myTimer() {
  const date = new Date();
  document.getElementById("horas").innerHTML = date.toLocaleTimeString();
}

function myStopFunction() {
  clearInterval(myInterval);
}
</script>

</body>
</html>
```

Element.classList

O Element.classListé uma propriedade somente leitura que retorna uma DOMTokenList coleção ativa das class atributos do elemento. Isso pode ser usado para manipular a lista de classes.

classList é uma alternativa conveniente para acessar a lista de classes de um elemento como uma string delimitada por espaço via element.className.

A DOMTokenList representando o conteúdo do class atributo do elemento. Se o class atributo não estiver definido ou vazio, ele retornará um vazio DOMTokenList, ou seja, um DOMTokenList com a length propriedade igual a 0.

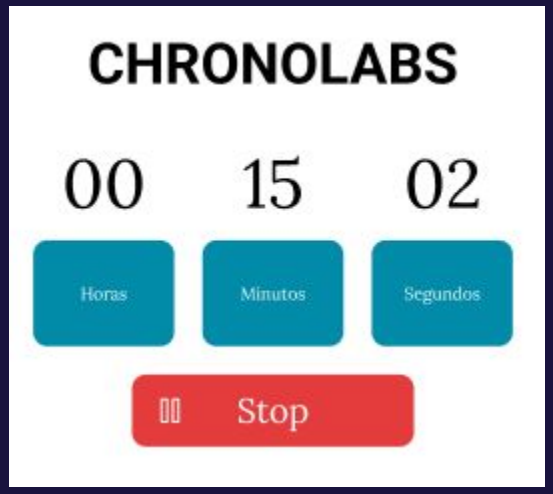
Embora a classList propriedade em si seja somente leitura, você pode modificar seu associado DOMTokenList usando os métodos add(), remove(), replace() e .toggle()

Teste classList.add()


```html
<!DOCTYPE html>
<html>
<style>
.adicionarColor {
  background-color: green;
  color:white;
  padding: 16px;
}
</style>

<body>

<button onclick="myFunction()">Adicionar</button>
<p>Clique no botão para adicionar o elemento CSS ao html.</p>

<div id="corPadolabs">
<p>Adicionei cor a PADOLABS</p>
</div>

<script>
function myFunction() {
  const list = document.getElementById("corPadolabs").classList;
  list.add("adicionarColor");
}
</script>

</body>
</html>
```

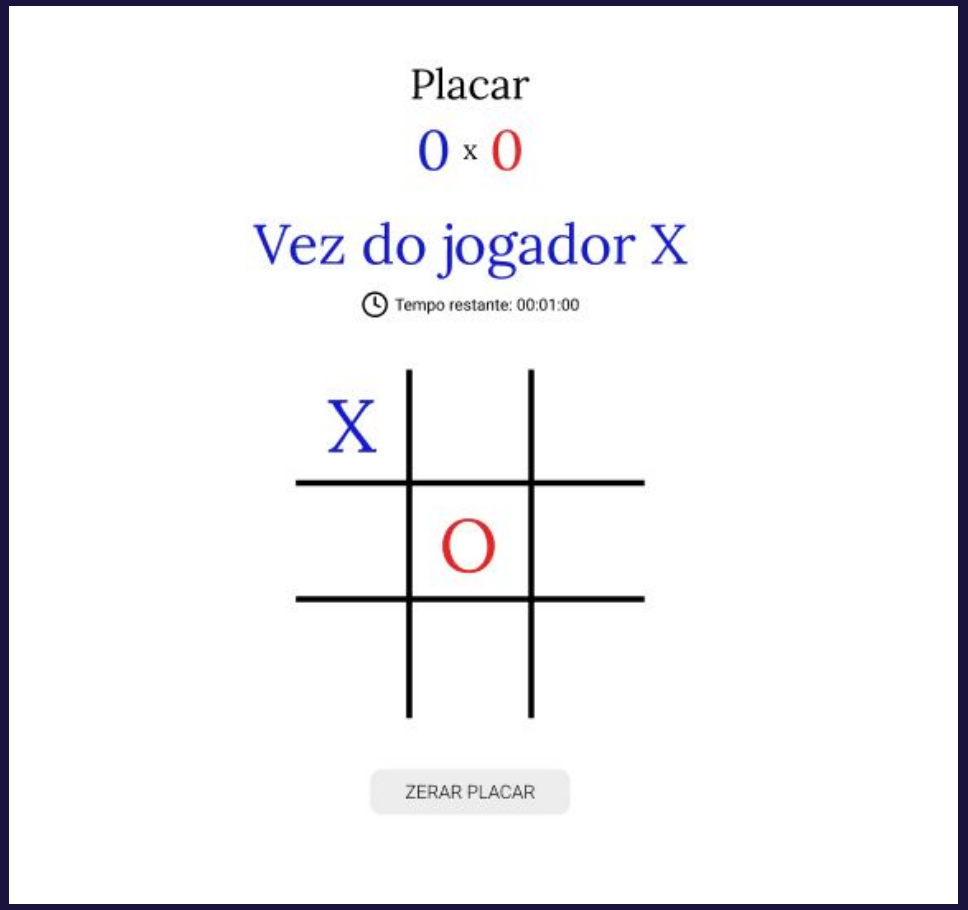
Exercício 1

Implemente um cronômetro conforme as imagens a seguir:



Exercício 2

Implementar o jogo da velha, conforme a tela abaixo:



Figma

Link para o figma com as telas:

https://www.figma.com/file/LyRiIOPksZ1W6YBDf4mHdq/Calculadora?node-id=0%3A1

Exercício 2

Regras:

- Utilizar a função alert(“”) para informar os o fim da partida (vencedor ou empate).
- Ao final da partida o vencedor ganha 1 ponto no placar.
- O botão “ZERAR PLACAR” volta os pontos para zero.
- Limitar o tempo da jogada em 1 minuto, caso o jogador não marque o campo desejado nesse tempo, conceder a vitória ao outro jogador.

Referências

1. https://www.caelum.com.br/apostila/apostila-html-css-javascript.pdf
2. https://www.scriptbrasil.com.br/download/apostila/837/
3. https://developer.mozilla.org/en-US/docs/Web/API
4. https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API#canvas_and_image_interfaces
5. https://developer.mozilla.org/en-US/docs/Web/API/Element/classList
6. https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement

PADO

Labs