

TRYBE

Modulo I - Introdução ao Desenvolvimento Web

Bloco 4 – Javascript

Refêrencia completa para HTML, CSS e JS: <https://www.w3schools.com/> .

1) Introdução

Let, var, const const para valores que não serão alterados / let e var com diferença de escopo de blocos, geralmente preferir let.

Tipos primitivos String, number, boolean, undefined, null. String + string é concatenada.

Tipagem dinâmica com *typeof*

Operadores aritméticos

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
=	x = y	x = x y
**=	x **= y	x = x ** y

Operadores de comparação

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

Operadores lógicos

e && , ou ||

Condicionais

```
if (condition1) {  
  // block of code to be executed if condition1 is true  
} else if (condition2) {  
  // block of code to be executed if the condition1 is false and condition2 is true  
} else {  
  // block of code to be executed if the condition1 is false and condition2 is false  
}
```

Alternativa quando não tiver else if: o **ternário**

let result = (condition) ? (execute A) : (or else execute B);

```
switch(expression) {  
  case x:  
    // code block  
    break;  
  case y:  
    // code block  
    break;  
  default:  
    // code block  
}
```

Diferença entre os dois: switch é mais interessante de usar quando tiver mais volume de condições.

Trabalhar no VsCode

Baixar Node js

Instalar extension Code Runner

Usar o Debug

Comentários no documento .js

// de uma linha (pode também selecionar e ativar no VSCode com ctrl k c, desativar com ctrl k u)

/* de mais do que uma linha */

Dicas diversas

Imprimir duas var ao mesmo tempo:

```
console.log(var1, var2);
```

--

Adaptar o jeito que foi escrito o input:

```
.toLowerCase() , .toUpperCase()
```

--

Decidir do nº de decimais (aqui 2):

```
console.log(lucro.toFixed(2))
```

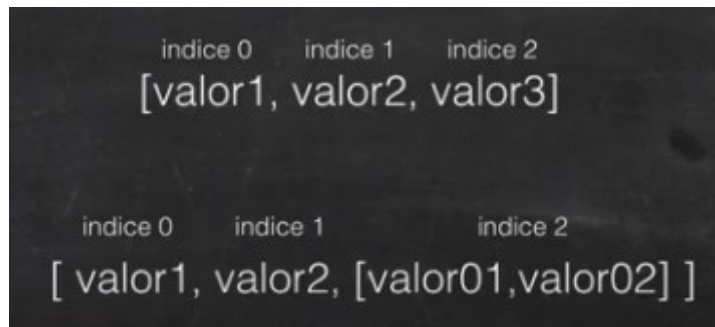
2) Array e loop for

ARRAY

Definição

Arrays são estruturas utilizadas para armazenar múltiplos valores em uma única variável.

Fluxo do funcionamento de array (pode ter array dentro de array):



Array.length para saber quanto items tem dentro.

Name-var-array[num] para chamar elemento com indice (sempre começa com 0).

Name-var-array [num1] [num2] para chamar no caso de array dentro de array.

Name-var-array [name-other-var] [num] também possível.

Toda syntax de array https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array

.length saber quantos itens tem o array

.slice (1,2) tirar elementos dando o indice deles

.sort() organizar em ordem alfabética e/ou numérica

.push para adicionar um elemento no final

ou

nome-var-array[nº] = adicionar elemento numa posição nº

.length-1 para saber last position (-2, -3 etc também existe para percorrer desde o fim).

FOR

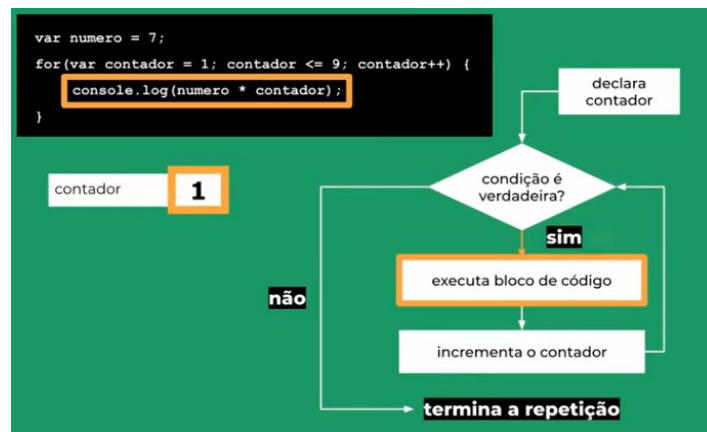
Estrutura de repetição.

Syntax

```
let cars = ["Saab", "Volvo", "BMW"];
for (let i = 0; i < cars.length; i++) {
  console.log(cars[i])
}
```

ou seja: “para(onde inicializar ciclo; até quando repetir pois vai acontecer sempre que essa condição for verdadeira; o que o contador evolui quando repetir) { fazer isso a cada iteração}”

Fluxo



Para passar todos os elementos:

```
var fruits = ["Apple", "Banana", "Orange"];  
for (x of fruits) {  
  console.log(x);  
}
```

break para quebrar a estrutura de repetição

Outros tipos de loops: while, do-while, for in.

Dicas diversas

let var = 0; let var = [0]; let var = ""; no começo, para deixar var declarada e preenchível ao longo da execução do código.

--

i=i+1 equivalente a i+=1 equivalente a i++ . O code climate prefere o segundo.

--

For dentro de for quando precisar iterar cada parte de dois arrays (exemplo megasena).

—

Cuidado com localização dentro e fora do for das declarações de variáveis, segundo o uso que quer delas, ou do console.log também.

--

Algoritmo de ordenação Bubble Sorting

<https://www.w3resource.com/javascript-exercises/javascript-function-exercise-24.php>

2) Algoritmo e lógica de programação

Definição

Algoritmo é uma *sequência de comandos* para resolver um problema.

Lógica de programação é a *capacidade analítica para transformar grandes problemas em pequenos* e encadeados comandos lógicos. A lógica é aplicável a todas as linguagens.

Baby Steps

1) Interpretar 2) Criar algoritmo 3) Codificar o algoritmo

4) Perguntas a serem feitas para avaliar o algoritmo:

Resolveu? Outras maneiras? Eficiente? Retirar ou trocar passos? Computador entendeu tudo?

3) Objetos e funções

OBJETOS

Syntax

```
{  
  indice: valor,  
  indice: valor  
}
```

```
let car = {  
  type: "Fiat",  
  model: "500",  
  color: "white",  
};
```

Conjunto *indice+valor= propriedade*.

Como acessar objeto:

`console.log(object.key)` ou `console.log(object["key"])`; (ou seja acessar a propriedade chave e não a string que existe dentro de chave).

Acessar a string : `object[key]`.

Como alterar desde fora o objeto:

- Adicionar propriedade: `object.key = "new value"` ou `object["key"] = "new value"` ;
- Remover propriedade: `delete.key` ;
- Selar um objeto para que não possa ser alterado assim: `object.seal(objectvarname)` .

FOR/IN

For-in é uma variação de laço for que garante facilidade ao lidar com objetos.

Syntax

```
for (let key in object){  
  something that you want to happen  
}
```

```
let car = {  
  type: "Fiat",  
  model: "500",  
  color: "white",  
};  
  
for (let i in car) {  
  console.log(i, car[i]);  
}
```

Com Array:

```
let cars = ["Saab", "Volvo", "BMW"];  
  
for (let i in cars) {  
  console.log(cars[i]);  
}
```

FUNÇÕES

Definição


Bloco de código que encapsula instruções que executam uma tarefa específica.

Um programa é feito de várias funções, cada uma com apenas uma responsabilidade.

Syntax

```
function nomeDaFuncao() {  
  // código que faz alguma  
  // coisa  
}
```

Parâmetros



```
function nomeDaFuncao(param1, param2) {  
  // código que faz alguma coisa  
}
```

```
function nomeDaFuncao(param1, param2) {  
→ var resultado = param1 + param2;  
  return resultado;  
}
```

Invocar: console.log(nomeFunction()) .

Method + return = Function

Uma function sempre tem um return, caso contrário, é apenas um método.

Premissas a serem respeitadas:

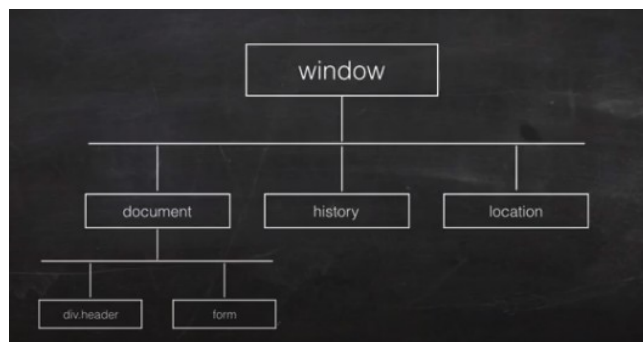
- nunca colocar na function de modificar um parâmetro;
- pelo mesmo parâmetro sempre deve dar o mesmo resultado.

4) DOM e Seletores

DOM (Document Object Model)

Ou seja, aula sobre como o HTML e o JavaScript se comunicam.

Ex de árvore DOM:



“A página HTML/CSS/JS que você faz é um programa. O navegador é quem interpreta esse código e, a partir dele, gera a página que você vê na Internet: o DOM é uma estrutura da sua página que o navegador monta quando lê. O seu intuito é justamente permitir ao programa acessar os elementos da página usando código e dar a ele o poder de manipulá-las.”

O DOM foi projetado para ser **independente de qualquer linguagem** de programação específica, disponibilizando a **representação estrutural do documento** a partir de uma única API consistente.

Seletores de elementos

Opções de lugares onde fazer a escritura do script:

- num arquivo .js separado e linkado no arquivo html
- com tag <script> no final do head do html
- com tag <script> no final body do html, melhor prática de renderização para seguir ordem lógica na hora do browser carregar a página.

getElementById

console.log(**document.getElementById("id")**) para ver div inteira

console.log(document.getElementById("id").**innerHTML**) para ver apenas texto contido na div

getElementsByName

console.log(document.getElementsByName("class")[**nºíndice**])

Cuidado: sem ;!

getElementsByTagName

console.log(document.getElementsByTagName("tag")[**nºíndice**].**innerHTML**) para ter o valor dentro dessa tag (por exemplo button)

querySelector

Simplifica os três precedentes em um.

document.querySelector("# ou . ou **nometag**")

Ele retorna apenas o primeiro elemento. Mesmo com id, os três anteriores têm melhor semântica.

querySelectorAll

Retorna uma array com todos os elementos que "casem" com a sua seleção, ao invés de retornar apenas o primeiro deles.

* Vantagem dos query: acessar todos tipos de CSS selectors. Mais flexibilidade mas menos performance do que ClassName e TagName.

* Diferença: retornam uma NodeList, Class e Tag retornam uma HTMLCollection. (HTML são apenas elementos HTML na página. NodeList pode ser outro elemento.)

Exemplo de syntax para ALTERAR elementos via selector e JS:

```
var paragraph = document.getElementById("paragraph");  
paragraph.style.color = "red";
```

ou .innerHTML para colocar texto dentro.

Para Adicionar ou alterar class de um elemento

.className = "";

6) Trabalhando com elementos

Buscando elementos

Por NODE	Por ELEMENTOS (html)
parentNode: retorna o elemento pai.	
childNodes: retorna um array com todos os elementos filhos	children
firstChild: retorna o primeiro filho	firstElementChild
lastChild: retorna o último filho	lastElementChild
nextSibling: retorna o próximo nó.	nextElementSibling: retorna o próximo elemento.
previousSibling: retorna o nó anterior.	previousElementSibling: retorna o elemento anterior.

Diferença entre Node e Element:

Node sempre é um element, mas element nem sempre é node. Element é a opção mais conservadora, é todo elemento html, ou seja com tag. O node pode ser por exemplo um texto solto.

Criando elementos

createElement

let vardeclarada = document.createElement("p"); (ou "div" ou elemento html)

Dar atributo para o elemento criado

vardeclarada.innerHTML = "";

.id = "";

.className = "";

img.src = "url";

.setAttribute

insertBefore

E como adicionar esses elementos a uma estrutura HTML já existente:

appendChild

Opções de syntax:

- com body pai é possível escrever document.body.appendChild();
- document.querySelector("classelementopai").appendChild(vardeclarada);
- varelementopai.appendChild(vardeclarada);

Removendo elementos

.removeChild

Syntax:

document.getElementById("iddoelementopai").removeChild(varfilhoquequerdeletar);

Dicas diversas

.join junta array e retorna uma string.

--

Iterar um elemento:

```
function countChildNodes(parent) {  
  let count = 0;  
  for (let child = parent.firstChild; child != null; child = child.nextSibling) {  
    count++;  
  }  
  return count;  
}
```

--

document.querySelector(".circle .small .yellow") para especificar com varias classes.

--

Dica: Cuidado não confundir var e class de um elemento quando escreve o código, erro frequente.

7) Eventos

Funcionalidade poderosa do JavaScript de “reagir” a eventos que acontecem em uma página web. Vc recebe ação do usuário sobre um elemento HTML e reage a esse evento, como um **estímulo de resposta**.

Dicas: - **Todos elementos** podem receber eventos: caixas de texto, botões e até mesmo elementos estáticos, como div e p. - Pode colocar **quantos event listeners** quiser em um mesmo elemento.

on load

Indica quando a página é carregada. É boa prática colocar todo seu script nele.

Sintax

```
window.onload=function(){  
  console.log("página carregada!");  
}
```

AddEventListener

Sintax

Aqui tem duas escolhas:

a) Com function anônima

```
let elementvarname = document.getElementById("idname");  
elementvarname.addEventListener('evento',function(){  
  ação reativa desejada, por exemplo alert("");  
})
```

b) Com function nomeada

Fluxo de escritura comum:

- on load
- declarar var para ter elemento chamável
- criar functions a serem aplicadas reativamente
- finalizar com addEventListener com syntax `varElemento.addEventListener("event",function)`

Aqui não tem () no final da function porque passa como parâmetro e não chama a function:

`botao.addEventListener('mouseover', abreCaixa);`

→ **Diferença entre opções a) e b), anônima ou não:** nenhuma ao nível da execução. Mas quando ela ficar sem nome é impossível de chamar depois, por exemplo para remover.

Eventos

Mais comuns são: **click, change, mouseover e keyup.**

Lista completa: https://www.w3schools.com/jsref/dom_obj_event.asp .

Eventos de timing: https://www.w3schools.com/js/js_timing.asp .

Dicas diversas

`var.remove()` para tirar aquele objeto que tem essa var

--

`var.value` para retornar valor contida naquela var

https://www.w3schools.com/jsref/prop_text_value.asp

--

Acessar o elemento html fruto de um evento: `event.target`

Detalhando aqui:

Solução para ter elemento livre com o qual associar um evento, sem ter que chamar especificamente, tendo acesso a todo o que o usuário tiver debaixo do mouse: `evento.target` . Pode escolher outro nome que "evento", é posicionado como param da function que é o segundo param do `addEventListener`. É o Browser que o entrega, sendo o elemento html presentemente debaixo do mouse.

Depois, isso tem opções específicas predefinidas, uma delas sendo o `target`.

O `evento.target` é a html de origem do evento.

--

Sobre **hoisting** e ordem do código: https://www.w3schools.com/js/js_hoisting.asp

--

`parseInt()` pega um valor string e retorna um numero inteiro.

(https://www.w3schools.com/jsref/jsref_parseint.asp

Diferença com **Number**: <https://medium.com/@iaforek/difference-between-number-and-parseint-in-javascript-77f3ae9b735b> .

--

`Math.random()` gera aleatoriamente valor entre 0 e 1.

Outros math objects praticos: https://www.w3schools.com/js/js_math.asp .

--

`img.style.transform="scaleX(1)";` //para frente

`img.style.transform="scaleX(-1)";` //para tras

8) Web storage

Diferença entre cookies e web storage

Em regra geral, utilizamos Cookies quando precisamos dos dados no cliente (browser) e no servidor. Caso contrário, utilizamos localStorage e sessionStorage que são mais seguros, mais recentes (apareceram com HTML5) e com maior limite de armazenamento.

COOKIES

Conceito e como manipular:

https://www.w3schools.com/js/js_cookies.asp

<https://developer.mozilla.org/pt-PT/docs/DOM/document.cookie>

Local e Session Storage

localStorage - salva os dados com armazenamento local no browser e *sem data de expiração*. Os dados não serão removidos quando o browser for fechado, ou seja, eles ficarão disponíveis enquanto não forem explicitamente removidos.

sessionStorage - salva os dados *apenas para a sessão atual*. Os dados são removidos assim que a pessoa fecha a aba (tab) ou o browser. Exemplo de uso: login de bancos, segurança alta.

Manipular

- Dois jeitos de criar chave:

```
localStorage.setItem("chave", "valor");
```

(onde chave é como o nome do campo e valor o que quero armazenar nele)

OU

```
localStorage.chave = "valor";
```

- Modificar valor:

Simplesmente reescrever o setItem acima com a mesma chave e mudando o valor.

- Fazer leitura dessas informações (e armanezar elas com var):

```
let valor = localStorage.getItem("chave");
```

OU

```
let valor = localStorage.chave ;
```

- chamar chave ou valor:

console.log(key) para chamar chave

console.log(localStorage[key]) para chamar valor

- Dois jeitos de remover especificamente:

```
delete localStorage.chave;
```

OU

```
localStorage.removeItem("chave");
```

- Remover todo:
localStorage.**clear()**;

Dicas diversas de uso

Verificar o armazenamento no browser chrome: Inspect – **Application**

–

Fluxo de escritura:

- a) preparar html para pegar preferências do user
- b) permitir a interação via listener
- c) armazenar o precedente com storage.