

Introdução ao JavaScript – parte 1

Profª Mª Denilce Veloso
denilce.veloso@fatec.sp.gov.br
denilce@gmail.com

JavaScript

JavaScript é uma linguagem de programação baseada em *scripts* e padronizada pela ECMA International (associação especializada na padronização de sistemas de informação).

JavaScript

- Criado na década de 90 por Brendan Eich a serviço da Netscape.
- O navegador mais popular dessa época era o Mosaic, da NCSA.
- Microsoft criou, em Agosto de 1996, uma linguagem idêntica para ser usada no Internet Explorer 3.
- Netscape decidiu normatizar a linguagem através da organização ECMA International, companhia que era especializada em padrões e normativas.
- Trabalhos em cima da normativa ECMA-262 começaram em Novembro de 1996. O nome composto por ECMA e JavaScript foi usado, resultando em ECMAScript.
- Mesmo com esse nome, até hoje a linguagem é conhecida carinhosamente por JavaScript.

JavaScript é a mesma coisa que ECMAScript??

- Sempre que há uma nova atualização dos recursos do JavaScript, ela é lançada pelo ECMAScript (abreviação ES). Dessa especificação que vieram siglas como ES6, ES2015, ES2020, ES2021, ES2022, ES2023 e ES2024.
- Pode-se dizer que ECMAScript é uma especificação de linguagem, ou seja, ela define os padrões para uma linguagem de programação, e o JavaScript é a implementação desses padrões.

JavaScript

Embora JavaScript e ECMAScript sejam geralmente usadas como sinônimos, JavaScript é mais do que está definido em ECMA-262:

- ✓ O **núcleo (core)** – *ECMAScript (sintaxe, instruções, etc)*;
- ✓ O *Document Object Model (DOM)*;
- ✓ O *Browser Object Model (BOM)* (objeto window)



Norma ECMA-262 define ECMAScript como uma base sobre a qual linguagens de script mais robustas possam ser construídas.

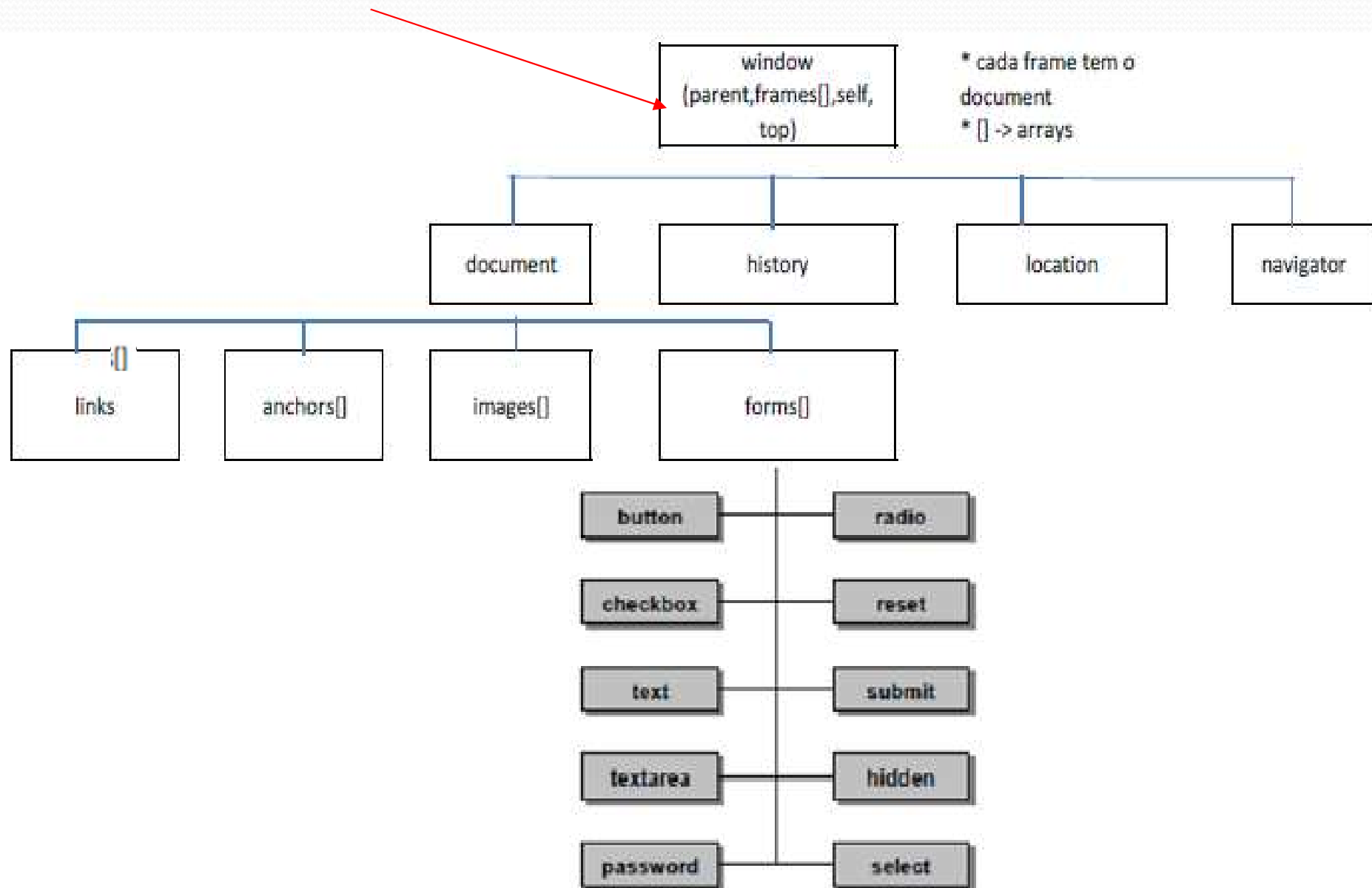
Slide 5

DDAOVO

Muitas linguagens de programação, incluindo o JavaScript, foram criadas com base na especificação ECMAScript, e as implementações da linguagem geralmente são compatíveis com essa especificação. Além disso, outras linguagens de script, como TypeScript, Dart e CoffeeScript, são projetadas para serem compiladas ou transpiladas para ECMAScript, o que significa que elas também seguem a especificação padrão da linguagem.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2023-04-02T14:37:36.375

JavaScript – BOM (Browser Object Model)



JavaScript

- **BOM** - *controle sobre a estrutura da página e conteúdo de forma hierárquica*

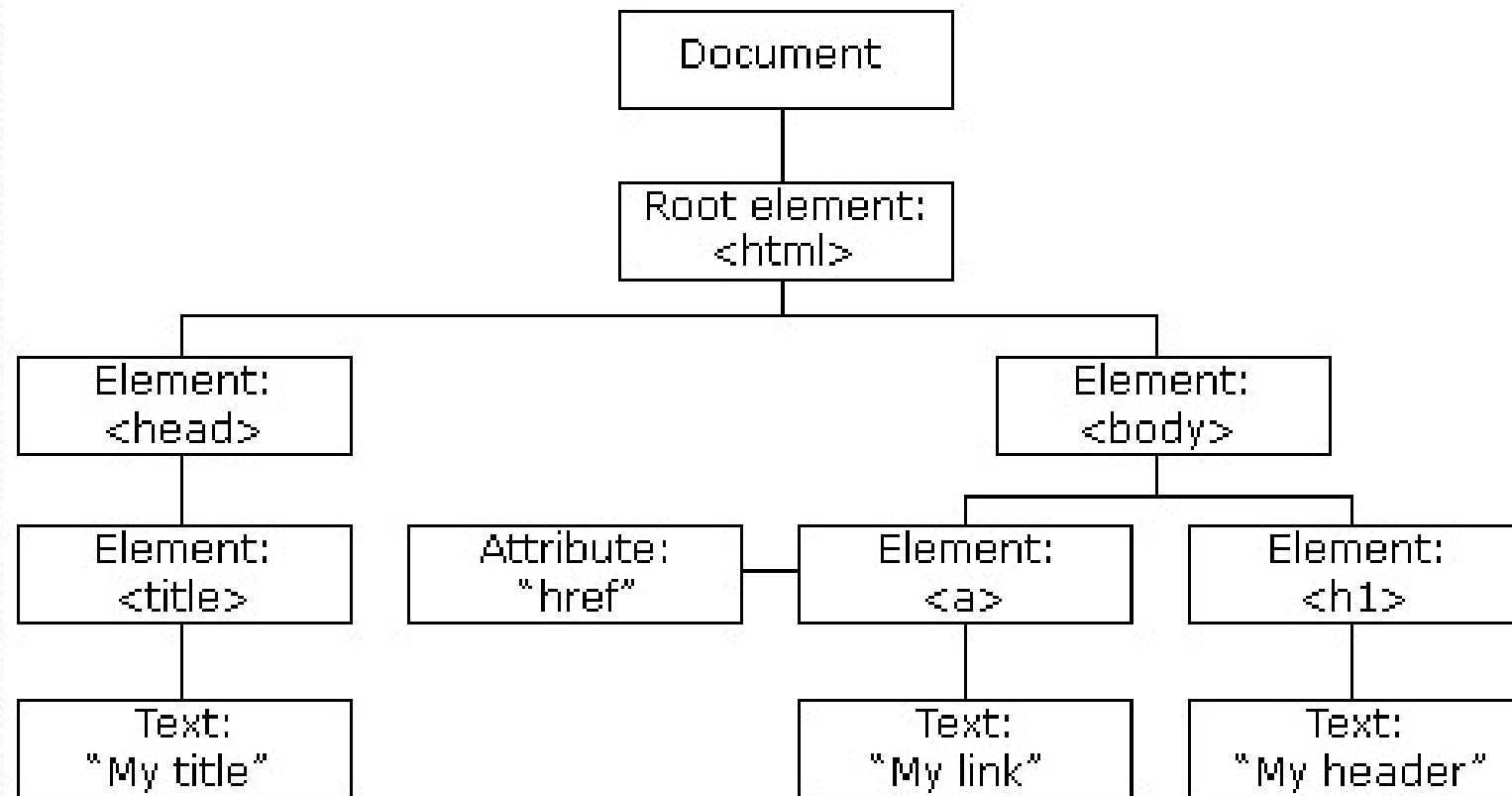
```
// Abre uma nova janela do navegador  
window.open("http://www.exemplo.com", "_blank");
```

```
// Redireciona para outra página  
window.location.href = "http://www.exemplo.com";
```

```
// Armazena um valor na sessão do usuário  
sessionStorage.setItem("username", "exemplo");
```

```
// Obtém a largura da tela do usuário  
var largura = screen.width; // ou let
```


JavaScript – DOM (Document Object Model)



JavaScript

- **DOM (Document Object Model)** - Representa a estrutura da página web carregada no navegador. Estrutura do documento HTML e XML em forma de árvore, onde cada nó representa um elemento do documento. Ex.: um parágrafo, um input, uma imagem etc.

// Obtém os valores dos campos do formulário

```
var nome = form.elements["nome"].value; //ou let
```

```
var email = form.elements["email"].value;
```

// Seleciona um elemento HTML na página e altera seu conteúdo

```
document.querySelector("#titulo").innerHTML = "Novo Título";
```

// Seleciona duas Div voltando um array

```
divs= document.querySelectorAll("div.classe1, div.classe2");
```

// Seleciona o campo de formulário “endereco”

```
var nomeInput = document.getElementById(“endereco”);
```


JavaScript

- ✓ Objetivo é adicionar interatividade a uma página Web;
- ✓ Processamento é feito na máquina do cliente (browser). O Node.js pôs um fim a essa questão, trouxe a linguagem para o lado do servidor colocando-a como concorrente das demais linguagens de servidor como PHP, Python, Ruby, ASP e etc. O Node.js usa a engine V8 do Google Chrome;
- ✓ Parecida com C;
- ✓ Apesar de conter Java no nome (*na época causou confusão*), a linguagem JavaScript é distinta da linguagem Java e apresenta recursos não disponibilizados em Java ou C++;

JavaScript

✓ Alguns autores consideram que ela não é puramente orientada a objetos, devido às diferenças em comparação com outras linguagens, Por exemplo: no JS, é possível criar objetos literal ou dinamicamente. **(a partir versão ES2 ECMAScript existe declaração formal classe);**

// Criar um objeto de maneira literal

```
var pessoa1 = {  
  nome: 'João'  
};
```

```
let pessoa2 = {  
  nome: 'Maria'  
};
```

//Criar um objeto sem declarar a classe

```
var pessoa = new Object();  
pessoa.nome = 'João';
```

ou

*** poderia usar uma função construtora “como se fosse a classe”*

JavaScript – Aplicações

- ✓ Criar janelas pop-up;
- ✓ Validação conteúdo nos formulários (tratamento de cliques de botões), fornecendo dicas, detecção de navegadores;
- ✓ Apresentar mensagens ao usuário;
- ✓ Realizar cálculos;
- ✓ Animações utilizando Api Canvas por exemplo;
- ✓ Pode ser utilizado no desenvolvimento Back-End com Node.JS;
- ✓ É utilizado em vários frameworks.

Slide 12

DDAOVO

A Canvas API provê maneiras de desenhar gráficos via JavaScript e via elemento HTML `<canvas>` . Entre outras coisas, ele pode ser utilizado para animação, gráficos de jogos, visualização de dados, manipulação de fotos e processamento de vídeo em tempo real. A Canvas API foca amplamente em gráficos 2D

DENILCE DE ALMEIDA OLIVEIRA VEL; 2022-04-06T16:03:07.133

JavaScript X Java

JAVA	JAVASCRIPT
Linguagem fortemente tipada	Linguagem fracamente tipada
Orientada a objetos	Suporte à orientação a objetos. Versões do ECMA a partir da versão 6, já tem definições de classes.
Processamento Síncrono	Processamento Assíncrono
A partir da v.8 apresenta algumas características da programação funcional	Implementa programação funcional

Processamento síncrono - processamentos ocorrem em sequência (sincronia). Os processos são executados em fila. É preciso que um processo termine para que outro processo seja executado.

Fonte :<https://medium.com/dev-cave/por-tr%C3%AAs-da-programa%C3%A7%C3%A3o-funcional-do-java-8-bd8cof172d45>

Programação funcional → <https://take.net/blog/devs/programacao-funcional-javascript>

Slide 13

DDAOV0

Para fazer a orientação a objeto ele utiliza Protótipos, que são o mecanismo pelo qual objetos JavaScript herdam recursos uns dos outros. Seria uma espécie de "modelo".

DENILCE DE ALMEIDA OLIVEIRA VEL; 2022-04-13T16:10:36.787

DDAOV0 0

Quando um objeto é criado sem uma classe, ele é criado com um protótipo padrão, que é o objeto Object.prototype.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2023-04-02T15:40:20.984

DDAOV0 1

Linguagens Closure e Haskell - puramente funcionais

Java, JavaScript, Rust, Golang, Scala, Python - tem características funcionais

DENILCE DE ALMEIDA OLIVEIRA VEL; 2024-03-23T02:47:29.673

Programação Funcional no JS

Programação funcional é o processo de **construir software através de composição de funções puras**, evitando compartilhamento de estados, dados mutáveis e efeitos colaterais. É declarativa ao invés de Imperativa.

Composição de funções é criar uma nova função através da composição de outras. Por exemplo, vamos criar uma função que vai **filtrar** um **array**, filtrando somente os números pares e multiplicando por dois:

```
const numeros = [2, 3, 4, 5, 6, 7, 8, 9, 10]

numeros.filter((numero) => numero % 2 === 0).map((numero) => numero * 2)
// [ 4, 8, 12, 16, 20 ]
```

filter – filtra
array
map –
percorre o
array

Slide 14

DDAOVO

é um princípio que incentiva os desenvolvedores a criar produtos ou serviços que possam ser utilizados por pessoas com diferentes capacidades e dispositivos como desktops, tablets ou smartphones.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2023-04-02T14:58:06.626

DDAOVO 0

Recursos do JavaScript que facilitam a programação funcional:

Funções de seta (arrow functions): Sintaxe concisa para escrever funções.

Métodos de array (map, filter, reduce): Permitem manipular arrays de forma declarativa.

Closures: Permitem que funções acessem variáveis de seu escopo léxico. (será falado adiante)

Promises e async/await: Facilitam a programação assíncrona de forma funcional. (será falado adiante)

Funções Puras - dado os mesmos argumentos, elas sempre retornarão o mesmo resultado, e não têm efeitos colaterais (não alteram variáveis fora de seu escopo).

Imutabilidade:

As funções não modificam os valores de entrada. Elas apenas os usam para calcular um novo valor.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2025-03-24T22:32:04.791

Programação Funcional no JS

*Exemplo: uma função usa como parâmetro outra função
(callback)*

```
const add = (x, y) => x + y;  
const multiply = (x, y) => add(x, y) * 2;  
console.log(multiply(2, 3)); // 10
```


JavaScript – Uso do ; (ponto e vírgula)

✓ Muitas vezes o comando é aceito sem ; (ponto e vírgula). Mas se código for colocado em uma ferramenta que retira espaços em branco, quebras de linha, etc., poderão ocorrer problemas, pois ele deixará todos os comandos em uma linha só. Ou em casos mais específicos, poderá não chegar ao resultado desejado.

Então, é considerada uma boa prática colocar o ";".

** Ver <http://loopinfinito.com.br/2013/10/22/mamilos-pontos-e-virgulas-em-js/>

JavaScript – Espaços em branco e quebras linha

✓ Quebras de linhas e espaços em branco, quando inseridos entre nomes de variáveis, nomes de funções, números e entidades similares da linguagem, são ignorados. Exemplos:

✓ As duas sintaxes a seguir são válidas:

`a=27; e a = 27;`

`alert("Olá") e alert("Olá")`

`function(){...} e function(){...}`

✓ As sintaxes a seguir causam um erro no script:

`a = 27; // espaço entre os algarismos de um número não é permitido`

`document.write("<p> Eu sou
uma string</p>")`

JavaScript – Espaços em branco e quebras linha

✓ A sintaxe a seguir é **válida**:

```
document.write("<p> Eu sou \\  
uma string</p>")
```

✓ A sintaxe a seguir causa um erro no script:

```
document.write \\  
("<p> Eu sou uma string</p>")
```

✓ As sintaxes a seguir são **válidas**:

```
document.write  
("<p> Eu sou uma string</p>")
```

```
document.write(`<p> Eu sou uma  
string</p>`);
```


JavaScript – Exemplo: UsandoPontoVirgula.html

```
<!DOCTYPE html>
<head>
<script>
    function foo1() {
        // ...
        return 'Uma string muito grande que não cabe na linha de cima'
    }
    // é o mesmo que
    function foo2() {
        // ...
        return;
        'Uma string muito grande que não cabe na linha de cima';
    }
    var x = foo1();
    alert(x);
    var x2 = foo2();
    alert(x2);
</script>
</head>
<body>
</body>
</html>
```

Imprime “Uma string muito grande ...

Imprime Undefined

**** Para concatenar texto usar operador +**

JavaScript – Formas de Inserção

Uma tag <script/> pode ser definida na seção **head** ou numa seção **body**. Na seção head, os scripts são executados quando são chamados ou quando algum evento ocorre. Na seção body, os scripts são executados na carga da página web.

E também pode ser definida **externamente**, Para definição externa, um arquivo com extensão “.js” precisa ser fornecido com as funções necessárias. (Essa é a forma mais comum, pois na vida prática esse arquivo terá muitas linhas)

JavaScript – Formas de Inserção

Uma prática bastante utilizada é colocar um `<script>` no final do body, permitindo que o conteúdo antes dele já **seja visualizado sem ter de esperar sua execução**. A desvantagem é que - se o seu script modifica significativamente o conteúdo e/ou sua apresentação e funcionalidade - será mostrada na página "estranha" e "mal formatada". Ou mesmo se um script muda o comportamento de um link ou botão, por exemplo, clicar nos mesmos antes do script executar causará um comportamento incorreto.

Conclusão: Conforme a necessidade o desenvolvedor pode escolher onde é o melhor lugar para se colocar o script.

JavaScript – UsandoInlineBody.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Primeira Página JavaScript</title>
</head>
<body>
  <script> document.write("Está é uma forma de usar JavaScript inline no
body")</script>
</body>
</html>
```


JavaScript – UsandoEmbutidoHead.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Primeira Página JavaScript</title>
  <script> document.write("Está é uma forma de usar JavaScript embutido no
head")</script>
</head>
<body>

</body>
</html>
```

**** Incomum esse tipo de utilização, se ocorre um erro aqui não carrega o resto da página. Vamos utilizar nos nossos exemplos só para facilitar a visualização de testes simples ☺**

JavaScript – UsandoExterna.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Outra forma de Utilizar JavaScript - Arquivo Externo</title>
  <script src="UsandoExterna.js"></script>
</head>

<body>

</body>

</html>
```



JavaScript – UsandoExterna.js

```
alert("Boa noite");
```


JavaScript – Async e Defer

```
<script src="script1.js" async></script>  
<script src="script2.js" async></script>  
````
```

script1.js` e `script2.js` serão baixados em paralelo com a página e executados assim que estiverem prontos, em qualquer ordem (são independentes).

```
<script src="script1.js" defer></script>
<script src="script2.js" defer></script>
````
```

- baixados em paralelo, mas adia a execução do script até que todo o documento HTML seja completamente analisado. Os scripts `defer` são executados na ordem em que aparecem na página.

JavaScript – Async e Defer

- Use `async` quando tiver scripts independentes que podem ser executados em qualquer ordem.
- Use `defer` quando desejar garantir que os scripts sejam executados na ordem correta após o carregamento completo do HTML.

Lembre-se de que é uma boa prática incluir seus scripts no final do documento ``<body>`` sempre que possível, independentemente de usar `async` ou `defer`, para garantir um carregamento mais rápido da página e evitar bloqueios de renderização.

JavaScript - Comentários

É possível usar dois tipos diferentes de comentários.

✓ As barras duplas (//) comentam uma linha:

```
// Esta é uma linha comentada  
var i = 1; // este é um comentário de linha
```

✓ Delimitadores de abertura e fechamento /* e */ marcam um bloco de comentários:

```
/* Este é um comentário com mais de uma linha.  
Comentários são muito úteis */
```




Primeiro Exercício de JavaScript

- Criar sua primeira página utilizando JavaScript

Padrões no JavaScript

- **Nome dos arquivos**

Utilizar extensão .js `nome-do-arquivo.js`.

- **Funções - utilizar CamelCase**

`function getNome()`

`function verificarCpf()`

- **Parâmetros - utilizar CamelCase**

`function setNome(nome)`

`function verificarCliente(codigoDoCliente)`

- **Variáveis - utilizar CamelCase**

`var nomeCliente;`

`var quantidadeDeProdutos;`

- <https://www.crockford.com/code.html>

Dicas de boas práticas no JavaScript

- Ver <https://code.tutsplus.com/pt/tutorials/24-javascript-best-practices-for-beginners--net-5399>

JavaScript - Variáveis

Variáveis dinâmicas podem ser criadas e inicializadas sem declarações formais. Existem dois tipos:

✓ **Global** - Declaradas/criadas fora de uma função. Podem ser acessadas em qualquer parte do programa.

✓ **Local** - Declaradas/criadas dentro de uma função. Só podem ser utilizadas dentro da função onde foram criadas e precisam ser definidas com a instrução `Var`.

Devem iniciar por uma letra, “_” ou \$, o restante da definição do nome pode conter qualquer letra ou número.

Case Sensitive: Cliente é diferente de cliente, que por sua vez é diferente de CLIENTE.

** Podem existir variáveis globais com o mesmo nome de variáveis locais, porém, esta prática não é aconselhável.

JavaScript - Variáveis

✓ **var** – Declara uma variável com escopo de função (funciona dentro ou fora do bloco)

✓ **let** – Declara uma variável com escopo de bloco (só funciona dentro do bloco declarado) → hoisting → `ReferenceError: Cannot access 'nomevariavel' before initialization`
(válido a partir do ES6)

✓ **const** – Declara uma constante (não pode mudar)

| keyword | const | let | var |
|-------------------|-------|-----|-----|
| global scope | NO | NO | YES |
| function scope | YES | YES | YES |
| block scope | YES | YES | NO |
| can be reassigned | NO | YES | YES |

Não tem escopo de bloco, porque é visto fora do bloco também, mas não é visto fora da função

DDAOVO

Então, em resumo, é uma boa prática usar `let` em vez de `var` em situações onde você deseja garantir que a variável tenha escopo de bloco, o que pode evitar erros devido a efeitos colaterais indesejados. Por outro lado, se você precisa definir uma variável que pode ser acessada em vários lugares dentro de uma função, pode ser preferível usar `var`.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2023-04-02T15:38:34.558

ATENÇÃO

Em projetos JavaScript modernos, é recomendado o uso de `let` e `const` devido às suas vantagens em relação a `var`.

Escopo de Bloco

Diferente de `var`, que tem escopo de função, `let` e `const` têm escopo de bloco (dentro do `if`, `for`).

Hoisting

Todas as declarações sofrem hoisting, mas `var` é inicializada com `undefined`, enquanto `let` e `const` permanecem não inicializadas, gerando um erro se acessadas antes da declaração (`ReferenceError`).

Redeclaração

`let` e `const` impedem a redeclaração (reatribuição) de variáveis no mesmo escopo, prevenindo erros.

Slide 34

DDAOV0

Então, em resumo, é uma boa prática usar `let` em vez de `var` em situações onde você deseja garantir que a variável tenha escopo de bloco, o que pode evitar erros devido a efeitos colaterais indesejados. Por outro lado, se você precisa definir uma variável que pode ser acessada em vários lugares dentro de uma função, pode ser preferível usar `var`.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2023-04-02T15:38:34.558

DDAOV0 0

Hoisting é um comportamento do JavaScript onde declarações de variáveis (com `var`, `let` e `const`) e funções são "elevadas" ao topo de seu escopo antes da execução do código. Isso significa que você pode usar uma função antes de declará-la no código.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2025-03-25T00:44:18.736

DDAOV0 1

Funções declaradas com `function` são elevadas completamente e podem ser chamadas antes da definição.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2025-03-25T00:46:04.633

JavaScript - Variáveis

Por convenção, identificadores ECMAScript utilizam o formato conhecido como *camel case* (*primeira letra minúscula e cada palavra adicional iniciando com uma letra maiúscula*).

Exs.: meuCaderno, suaCasa, primeiraVariavel.

É possível mudar o valor e também o tipo mas não é recomendado.

`var message = "Bom dia"; //string`

`message = 100; //número, não recomendado`

//se fosse usado `let message="Bom dia";` não permite redeclarar

`var` // É POSSIVEL DECLARAR VÁRIAS AO MESMO TEMPO

`nome = "Matheus",`

`idade = 30,`

`turma = "A";`

JavaScript – Regras de escopo para variáveis

- Variáveis declaradas com VAR dentro de uma função, não são “vistas” fora da função.
- Variáveis declaradas com VAR dentro de um bloco, por exemplo um if, são “vistas” fora do bloco.
- Variáveis usadas sem declarar VAR, se existir uma global com o mesmo nome, a variável vai ser assumida como a global.
- Variáveis declaradas localmente ou globalmente com VAR mas não inicializadas, ficam com valor indefinido (UNDEFINED).

JavaScript – Exemplo: UsandoVariaveis1.html(1)

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <meta charset="UTF-8">
  <title>Utilizando variáveis</title>

  <script>
    /* VARIÁVEIS GLOBAIS */

    var resultado;
    var resultado2; /* sem atribuição fica UNDEFINED */
    var resultado3;
    var resultado4;
    var resultado5 = "Conteúdo 5";
    var resultado6 = "Conteúdo 6";

    resultado = prompt("Qual é o seu nome?");
```


JavaScript – Exemplo: Usando Variaveis1.html(1)

```
var teste = function () {  
    var resultado6; /* local - undefined */
```

```
    resultado5; /* está local */
```

```
    resultado7 = "dentro da função"; /* não existe global com esse nome, SEM  
VAR fica acessível globalmente */
```

```
    resultado3 = "mudei dentro da função"; /* usando variável global */  
    resultado5 = "Conteúdo 5 na função";  
    return "resultado da função";  
};
```

```
resultado4 = teste(); /* atribuindo resultado da função */
```

```
resultado7 = "fora da função";
```


JavaScript – Exemplo: Usando Variaveis1.html(3)

```
alert("resultado=" + resultado + " \n resultado2=" + resultado2 + " \n  
resultado3=" + resultado3 + " \n resultado4 =" + resultado4 +  
" \n resultado5 =" + resultado5 + " \n resultado6 =" + resultado6 + " \n  
resultado7 =" + resultado7);
```

```
//bloco  
if (true){  
    var teste="conteúdo de teste";  
    let teste1="conteudo de teste1";  
    alert("dentro do bloco "+teste);  
    alert("dentro do bloco "+teste1);  
}  
alert("fora do bloco " + teste);  
alert("fora do bloco " + teste1); // nao vai mostrar
```

```
</script>  
</head>  
<body>  
</body>  
</html>
```


EXERCÍCIO - Escopo Variáveis - JavaScript

```
<script>
  x = 1;
  var y = 2;

  function exemplo() {
    var z = 3; // fica local
    w = 4;
    let i=5;

    x = x * 10;
    y = y * 10;
    return `x=${x} y=${y} z=${z} w=${w} i=${i}`;
  }
  alert(exemplo()); // resulta 10 20 3 4 5

  // externo - vai dar erro
  alert(`x=${x} y=${y} z=${z} w=${w} i=${i}`);

</script>
```


JavaScript – Escopo de Variáveis

O que será impresso?



DV0

```
if (true) {  
    var i = 50;  
}  
alert(i);
```


DV0

R. 50

Denilce Veloso; 2024-09-09T15:37:00.648

JavaScript – Escopo de Variáveis

O que será impresso?



```
if (true) {  
    let i = 50;  
}  
alert(i);
```


DV0

R. undefined

Denilce Veloso; 2024-09-09T15:38:22.065

JavaScript - Criando Variáveis

Existem três tipos de variáveis: Numéricas, Booleanas e Strings.

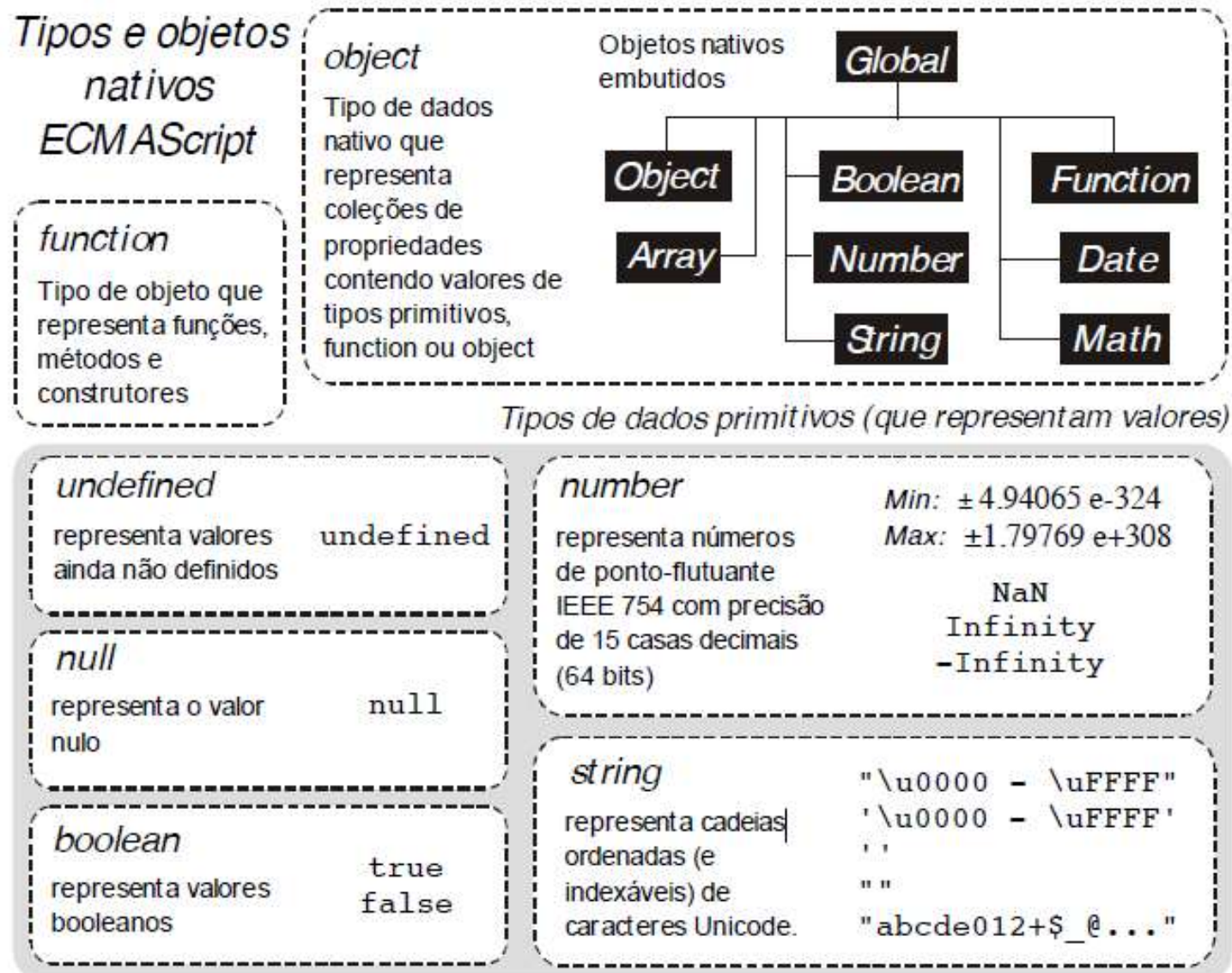
Numéricas - armazenam números;

Booleanas – armazenam valores lógicos (True/False);

Strings - sequência de caracteres.

As strings podem ser delimitadas por **aspas simples ' ou duplas "**. Se começar com as aspas simples, deve terminar com aspas simples, da mesma forma para as aspas duplas.

JavaScript - Criando Variáveis



**** apenas null e undefined não tem métodos, todos demais podem ser usados como objetos**

JavaScript - Variáveis String

Podem ser incluídos dentro de uma string alguns caracteres especiais, como `\b`, `\n`, `\\`, `\Xxx` (hexadecimal) etc

O JavaScript reconhece ainda um outro tipo de conteúdo em variáveis, que é o **NULL**. Na prática isso é utilizado para a manipulação de variáveis não inicializadas sem que ocorra um erro no seu programa.

Variável com valor NULL, significa dizer que ela possui um **valor desconhecido ou nulo, o null não é igual a nada, nem mesmo ao próprio null.**

JavaScript - Criando Variáveis - Exemplo: UsandoVariaveis2.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Utilizando variáveis</title>

  <script> /* globais */
var cidade = "SOROCABA";  /** aspas duplas ou simples **/
  var estado = 'SP';
  var nome1 = "\n Antonio"; /* passa para outra linha */
  var nome2 = "\n João";
  var nome3 = "\r Carlos"; /* return */
  var nome4 = null;
  var nome5 = "\\ Maria";
  var nome6 = "\xAE Bia";    hexa 2carac → ®
  var nome7 = "\u00AE Lia";  hexa 4carac → ®
  alert(cidade+nome1 + nome2 + nome3 + "\n" + nome4 + "\n" + nome5 + "\n" +
nome6 + "\n" + nome7 + "\n fim"); </script>
</head>
<body>
</body>
</html>
```


Unicode x UTF-8

Unicode é um sistema de codificação numérica que atribui um valor numérico único a cada caractere, enquanto UTF-8 é um esquema de codificação de caracteres que representa os caracteres Unicode em bytes. Em outras palavras, a **tabela Unicode define os códigos numéricos para cada caractere, enquanto UTF-8 é uma maneira de armazenar e transmitir esses caracteres**. Exemplos:

| Unicode | caractere | código UTF-8 | hexadecimal |
|---------|-----------|----------------------------|-------------|
| U+0021 | ! | 00100001 | 0x21 |
| U+0022 | " | 00100010 | 0x22 |
| U+002D | - | 00101101 | 0x2D |
| U+0039 | ' | 00100111 | 0x39 |
| U+0041 | A | 01000001 | 0x41 |
| U+0042 | B | 01000010 | 0x42 |
| U+0061 | a | 01100001 | 0x61 |
| U+0062 | b | 01100010 | 0x62 |
| U+007E | ~ | 01111110 | 0x7E |
| U+00C0 | À | 11000011 01000000 | 0xC380 |
| U+00E3 | ã | 11000011 10100011 | 0xC3A3 |
| U+00E7 | ç | 11000011 10100111 | 0xC3A7 |
| U+00E9 | é | 11000011 10101001 | 0xC3A9 |
| U+00FF | ÿ | 11000011 10111111 | 0xC3BF |
| U+03A3 | Σ | 11001110 10100011 | 0xCEA3 |
| U+03B1 | α | 11001110 10110001 | 0xCEB1 |
| U+2014 | — | 11100010 10000000 10010100 | 0xE28094 |

JavaScript - Criando Variáveis

```
let nomeCliente = "João Carlos"; //string
```

```
let alunoBolsista = True; //boolean
```

```
let intNum = 55; // inteiro
```

```
let octalNum1 = 070; // octal para 56 (precedido de o, usa o .. 7)
```

```
let hexNum1 = 0xA; // hexadecimal para 10 (precedido de ox, usa 0..9, A..F)
```

```
let binNum1 = 0b1010; // binário (precedido de ob, usa-se 0 e 1)
```

****** Para definir um número de ponto flutuante, basta incluirmos um ponto decimal e pelo menos um número após o ponto

```
let floatNum1 = 1.1;
```

```
let floatNum2 = 0.1;
```

```
let floatNum3 = .1; //válido, mas não recomendado
```

****** *ver UsandoVariaveis3.html*

JavaScript - Variáveis String

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
  <title>Testando IF</title>
  <script>
    let frase = "Boa noite, alunos!!!";

    alert("Comprimento=" + frase.length);

    alert("Caracter na posição 10>> " + frase.charAt(9)); /*, */

    alert("Fazendo replace de alunos>> " + frase.replace("alunos", "queridos"));

    alert("Tudo para Maiúsculo>> " + frase.toUpperCase());

    /** pegando da posição 0 até a posição 9 */
    alert("Pegando partes da frase>> " + frase.substring(0, 9));

    / / ** se não for especificado o segundo parâmetro, pega da posição inicial indicada até o fim **/

    alert("Procurando noite" + frase.lastIndexOf("noite"));
  </script>
</head>
<body>
</body>
</html>
```

**** Ver UsandoVariaveis4.html**

JavaScript - Variáveis Números

- ✓ O maior número é armazenado em `Number.MAX_VALUE` e é `1.7976931348623157e+308` na maioria dos navegadores .
- ✓ Se o resultado de um cálculo é um número que não pode ser representado pela faixa de valores de JavaScript, ele automaticamente recebe o valor especial `Infinity` (`-Infinity` para valores negativos). Função *isFinite()* testa se o valor é finito. Por ex. `isFinite(10/0)` → irá retornar `False`

JavaScript - Palavras Reservadas (1)

Palavras-chave JavaScript

| | | | |
|--------------|------------|--------|-------|
| break | else | new | var |
| case | finally | return | void |
| catch | for | switch | while |
| continue | function | this | with |
| default | if | throw | |
| delete | in | try | |
| do | instanceof | typeof | |

Palavras reservadas pela especificação ECMA-262

| | | | |
|----------|------------|-----------|--------------|
| abstract | enum | int | short |
| boolean | export | interface | static |
| byte | extends | long | super |
| char | final | native | synchronized |
| class | float | package | throws |
| const | goto | private | transient |
| debugger | implements | protected | volatile |
| double | import | public | public |

JavaScript - Palavras Reservadas (2)

Palavras reservadas dos dispositivos de hospedagem

| | | | |
|----------|----------|-----------|-------------|
| alert | eval | location | open |
| array | focus | math | outerHeight |
| blur | function | name | parent |
| boolean | history | navigator | parseFloat |
| date | image | number | regExp |
| document | isNaN | object | status |
| escape | length | onLoad | string |

JavaScript – Operadores Numéricos

Operadores numéricos são: +, -, *, / e % (resto).

Valores são atribuídos usando = e também há instruções de atribuição compostas como += e -=.

☺Cuidado:

```
let soma1 = "3" + 4 + 5;    -> 345
```

```
let soma2 = 3 + 4 + "5";   -> 75 (concatena)
```

```
let soma3 = 3 + "4" + 5;   -> 345
```

```
let numero1 = 23
```

```
let numero2 = "42"
```

```
let soma = numero1 + numero2 -> 2342 (concatena)
```


JavaScript – Operadores Numéricos - Exercícios



1) A partir do código abaixo, o que será impresso nos alerts?

```
<script>
```

```
let a = 3; let b = 6; let c = -3; let d = 4.5; let e = "5"; let f="6";
```

```
let resultado1 = ((b * c) + (a * 17) * b) - 2 * 2.5; alert(resultado1);
```

```
let resultado2 = e+3; alert(resultado2);
```

```
let resultado3 = (d > 2 ? 1 : 0)+b;
```

```
alert(resultado3);
```

```
let resultado4= 10; resultado4 += e; alert(resultado4);
```

```
let resultado5=(a===f? "saída a":"saída b"); alert(resultado5);
```

```
</script>
```

a) 283 53 7 105 saída b

b) 283 8 7 105 saída b

c) 715 53 7 105 saída b

d) 283 53 7 105 saída a

e) Nenhuma das anteriores

Slide 54

DDAOVO

R. letra a

DENILCE DE ALMEIDA OLIVEIRA VEL; 2025-03-24T15:48:45.770

JavaScript – Operadores Numéricos - Exercícios

2) O que será impresso no código abaixo?



```
let i = 1 + "2" + 3;
```

```
for (j = 0; j < 10; j++) {  
    alert("resultado=" + (i+j));  
}
```

3) E se colocar o mesmo alert **fora do for**, qual valor será impresso?

Slide 55

DDAOV0

R. 12310

DENILCE DE ALMEIDA OLIVEIRA VEL; 2024-03-23T02:22:29.385

DDAOV0 0

Tanto faz colocar $j=0$ ou $\text{var } j=0$

DENILCE DE ALMEIDA OLIVEIRA VEL; 2024-04-03T21:56:40.485

JavaScript – Operadores Numéricos



4) O que será impresso no código abaixo?

```
let i = 1 + "2" + 3;
```

```
for (j = 0; j < 10; j++) {  
    alert("resultado=" + (i+j));
```

→ colocar um if aqui e se for j=5 dar um break

```
}
```

Qual seria o resultado impresso em um alert aqui???

DDAOVO

R. 5

DENILCE DE ALMEIDA OLIVEIRA VEL; 2024-03-23T02:23:32.210

JavaScript – Operadores Numéricos

O que será impresso?



DV0

```
alert(8/2*(2+2));  
alert("8"/2*(2+2));  
alert("8/2"*(2+2));  
alert("8"/2*("2+2"));  
alert("4"+"3"*"2"-"1");
```

DV0

R.

16

16

NaN

NaN

45

Denilce Veloso; 2024-09-15T15:32:40.977

Curiosidade

```
if (0.5 + 0.1 == 0.6) {  
    alert('true');  
}  
else {  
    alert('false');  
}
```



DDA

```
if (0.2 + 0.1 == 0.3) {  
    alert('true');  
}  
else {  
    alert('false');  
    alert(0.2+0.1); /*0.30000000000000004*/  
    alert((0.2+0.1).toFixed(2))  
}
```

ponto flutuante

na segunda expressão, $0.2 + 0.1$, embora matematicamente isso seja 0.3, em JavaScript há uma imprecisão na representação de números de

DDAOVO

A questão da precisão dos números de ponto flutuante é uma característica das representações binárias desses números, que pode variar dependendo da implementação da linguagem e do padrão utilizado.

Para a primeira expressão, $0.5 + 0.1$, o resultado é 0.6. Neste caso específico, não há uma perda significativa de precisão durante as operações de soma e comparação, então o resultado é true.

Porém, na segunda expressão, $0.2 + 0.1$, embora matematicamente isso seja 0.3, em JavaScript há uma imprecisão na representação de números de ponto flutuante. Internamente, o valor armazenado para $0.2 + 0.1$ pode ser uma fração muito pequena menor ou maior do que 0.3, devido às limitações na precisão de ponto flutuante. Portanto, a comparação $0.2 + 0.1 == 0.3$ pode resultar em false.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2024-03-23T01:33:56.570

JavaScript – Operadores Unários

✓ Incremento de Prefixo e Sufixo (++)

Supondo:

```
let a=2;
```

`alert(++a + 2);` → soma 1 na variável a **antes** de somar 2 → imprime 5

`alert(a++ + 2);` → soma 1 na variável a **após** somar 2 → imprime 4

** questão momento da impressão

✓ Decremento de Prefixo e Sufixo (--)

`alert(--a + 2)` → subtrai 1 da variável a **antes** de somar 2 → imprime 3

`alert(a-- + 2)` → subtrai 1 da variável a **após** somar 2 → imprime 4

** questão momento da impressão

** *ver arquivo: UsandoOperadoresUnarios.html*

JavaScript – Operadores Comparação

| Operador | Descrição | Exemplos que retornam verdadeiro |
|------------------------------|--|---|
| Igual (==) | Retorna verdadeiro caso os operandos sejam iguais. | <pre>3 == var1 "3" == var1 3 == '3'</pre> |
| Não igual (!=) | Retorna verdadeiro caso os operandos não sejam iguais. | <pre>var1 != 4 var2 != "3"</pre> |
| Estritamente igual (===) | Retorna verdadeiro caso os operandos sejam iguais e do mesmo tipo. Veja também Object.is e igualdade em JS . | <pre>3 === var1</pre> |
| Estritamente não igual (!==) | Retorna verdadeiro caso os operandos não sejam iguais e/ou não sejam do mesmo tipo. | <pre>var1 !== "3" 3 !== '3'</pre> |
| Maior que (>) | Retorna verdadeiro caso o operando da esquerda seja maior que o da direita. | <pre>var2 > var1 "12" > 2</pre> |
| Maior que ou igual (>=) | Retorna verdadeiro caso o operando da esquerda seja maior ou igual ao da direita. | <pre>var2 >= var1 var1 >= 3</pre> |
| Menor que (<) | Retorna verdadeiro caso o operando da esquerda seja menor que o da direita. | <pre>var1 < var2 "12" < "2"</pre> |
| Menor que ou igual (<=) | Retorna verdadeiro caso o operando da esquerda seja menor ou igual ao da direita. | <pre>var1 <= var2 var2 <= 5</pre> |



JavaScript – Operadores Comparação

O que será impresso nos códigos abaixo?

```
if ("3"===3)
    alert("iguais");
else
    alert("diferentes");
```

```
let numero=20;
if (numero=="20")
    alert("iguais");
else
    alert("diferentes");
```



JavaScript – Operadores Lógicos

Logical operators

| Operador | Utilização | Descrição |
|----------|----------------|---|
| && | expr1 && expr2 | (E lógico) Retorna expr1 caso possa ser convertido para falso; senão, retorna expr2. Assim, quando utilizado com valores booleanos, && retorna verdadeiro caso ambos operandos sejam verdadeiros; caso contrário, retorna falso. |
| | expr1 expr2 | (OU lógico) Retorna expr1 caso possa ser convertido para verdadeiro; senão, retorna expr2. Assim, quando utilizado com valores booleanos, retorna verdadeiro caso ambos os operandos sejam verdadeiro; se ambos forem falsos, retorna falso. |
| ! | !expr | (Negação lógica) Retorna falso caso o único operando possa ser convertido para verdadeiro; senão, retorna verdadeiro. |

```
let a1 = true && true;
// t && t retorna true
let a2 = true && false;
// t && f retorna false
let a3 = false && true;
// f && t retorna false
let a4 = false && (3 == 4);
// f && f retorna false
```

```
let o1 = true || true; // t
|| t retorna true
let o2 = false || true; // f
|| t retorna true
let o3 = true || false; // t
|| f retorna true
let o4 = false || (3 == 4); //
f || f retorna false
```

```
let n1 = !true; //
!t retorna false
let n2 = !false; //
!f retorna true
```

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators

(parênteses)

A estrutura básica do **if** em **javascript** é:

```
if ( condição )  
{ código a ser executado }
```

Essa condição é feita por operadores de comparação como: **maior**, **menor**, **menor igual**, **igual**, **maior igual**, **diferente**. Também podem ser usados ou não em conjunto com os operadores lógicos: **e** (**&&**) , **ou**(**||**), etc.

JavaScript – Exemplo: Usandolf1.html

```
<!DOCTYPE html>  
<html lang="pt-br">
```

```
<head>  
  <meta charset="UTF-8">  
  <title>Testando IF</title>
```

```
  <script>  
    if (25 == 35) {  
      alert("Estes números são iguais");  
    } else {  
      alert("Estes números não são iguais");  
    }  
  
```

```
</script>  
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```


JavaScript – Exemplo: Usandolf2.html

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <meta charset="UTF-8">
  <title>Teste IF</title>

</head>

<body>
  <script>
    let name = prompt("Dia da Semana");

    if (name == "sexta") {
      name = "eba sexta!!";
    } else if (name == "segunda") {
      name = "ah segundona!!";
    } else {
      name = "Vamos " + name + "!!";
    }

    alert(name);
  </script>
</body>
</html>
```

JavaScript – Operadores Bit a Bit

| Operador | Expressão | Descrição |
|--|-------------------------------|---|
| AND | <code>a & b</code> | Retorna um 1 para cada posição em que os bits da posição correspondente de ambos operandos sejam uns. |
| OR | <code>a b</code> | Retorna um 0 para cada posição em que os bits da posição correspondente de ambos os operandos sejam zeros. |
| XOR | <code>a ^ b</code> | Retorna um 0 para cada posição em que os bits da posição correspondente são os mesmos.
[Retorna um 1 para cada posição em que os bits da posição correspondente sejam diferentes.] |
| NOT | <code>~ a</code> | Inverte os bits do operando. |
| Deslocamento à esquerda | <code>a << b</code> | Desloca a em representação binária b bits à esquerda, preenchendo com zeros à direita. |
| Deslocamento à direita com propagação de sinal | <code>a >> b</code> | Desloca a em representação binária b bits à direita, descartando bits excedentes. |
| Deslocamento à direita com preenchimento zero | <code>a >>> b</code> | Desloca a em representação binária b bits à direita, descartando bits excedentes e preenchendo com zeros à esquerda. |

Incrementa e inverte sinal

Exemplo: UsandoOperadores.html

JavaScript – Operadores Bit a Bit

Qual o resultado das operações?

$8 \wedge 2$

$4 \& 1$



Slide 67

DDAOV0

1000

0010

1010 • 8+2=10

DENILCE DE ALMEIDA OLIVEIRA VEL; 2022-04-13T16:16:18.082

JavaScript – Operadores Atribuição

| Nome | Operador encurtado | Significado |
|---|--------------------------------|-----------------------------------|
| Atribuição | <code>x = y</code> | <code>x = y</code> |
| Atribuição de adição | <code>x += y</code> | <code>x = x + y</code> |
| Atribuição de subtração | <code>x -= y</code> | <code>x = x - y</code> |
| Atribuição de multiplicação | <code>x *= y</code> | <code>x = x * y</code> |
| Atribuição de divisão | <code>x /= y</code> | <code>x = x / y</code> |
| Atribuição de resto | <code>x %= y</code> | <code>x = x % y</code> |
| Atribuição exponencial | <code>x **= y</code> | <code>x = x ** y</code> |
| Atribuição bit-a-bit por deslocamento á esquerda | <code>x <<= y</code> | <code>x = x << y</code> |
| Atribuição bit-a-bit por deslocamento á direita | <code>x >>= y</code> | <code>x = x >> y</code> |
| Atribuição de bit-a-bit deslocamento á direita não assinado | <code>x >>>= y</code> | <code>x = x >>> y</code> |
| Atribuição AND bit-a-bit | <code>x &= y</code> | <code>x = x & y</code> |
| Atribuição XOR bit-a-bit | <code>x ^= y</code> | <code>x = x ^ y</code> |
| Atribuição OR bit-a-bit | <code>x = y</code> | <code>x = x y</code> |

Exemplo: UsandoOperadores2.html

JavaScript – Caixas Diálogo: Alert, Confirm e Prompt

As caixas de diálogo podem ser: de alerta, de confirmação ou de prompt de entrada. Todas elas são chamadas de forma simples e intuitiva.

JavaScript – Caixas Diálogo: Alert, Confirm e Prompt

Alert - mostra apenas uma mensagem com um botão de confirmação para que esta seja fechada;

Prompt – requer uma entrada ao usuário e retorna um valor, normalmente não é muito utilizada porque são utilizados os inputs;

Confirm - também retorna um valor que pode ser true (verdadeiro) ou false (falso).

JavaScript – Caixas Diálogo- Antes de continuar

Abra mode debugger do Chrome (f12)

Opção console.

Digitar:

```
console.log("olá");
```

Ele imprime olá e dá undefined (significa que não está voltando nada de dados que possam ser salvos)

JavaScript – UsandoCaixasDialogo.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Testando Identidade</title>

  <script>
    alert("ALERT do JavaScript!");
    let nome;

    nome = prompt("Qual é o seu nome?");

    alert("Seu nome é " + nome);

    let decisao = confirm("Clique em um botão!");
    if (decisao) {
      alert("Você clicou no botão OK,\n retornou o valor: " + decisao);
    } else {
      alert("Você clicou no botão CANCELAR,\n retornou o valor: " + decisao);
    }
  </script>
</head>
<body>
</body>
</html>
```


JavaScript – Exercício para entrega

Utilize as caixas de diálogo: Alert, Prompt e Confirm da forma desejada. **Utilizar arquivo JS externo.**

Disponibilizar no GitHub →
seuusuario\PWEB\Atividade6

- 1) **Media.html** : Leia o nome e as quatro notas de um aluno. Calcule e mostre a média aritmética. (*parseFloat* → *converte string para ponto flutuante*)
- 2) **Operacoes.html**: Receba dois números e calcule e mostre:
 - ✓ a soma dos dois;
 - ✓ a subtração do primeiro pelo segundo;
 - ✓ o produto dos dois;
 - ✓ a divisão do primeiro pelo segundo;
 - ✓ o resto da divisão do primeiro pelo segundo.

JavaScript – Operadores Especiais

- ✓ operador condicional (?)
- ✓ operador vírgula (,)
- ✓ delete
- ✓ in
- ✓ new
- ✓ instanceof
- ✓ this
- ✓ typeof
- ✓ Void
- ✓ coalescência nula (??)

JavaScript – Operadores Especiais

O operador condicional (?) é o único operador JavaScript que utiliza três operandos. O operador pode ter um de dois valores baseados em uma condição.

A sintaxe é: `condicao ? valor1 : valor2`

Exemplo 1:

```
let fatec = "Sorocaba";
```

```
let semestre = 4;
```

```
let status = (fatec = "Sorocaba") && (semestre = 4) ? "Fatec Sorocaba - 4º  
Semestre" : "Outras Fatecs ou outro semestre";
```

```
alert("status=" + status); → "status=Fatec Sorocaba - 4º Semestre"
```


JavaScript – Operadores Especiais

Exemplo 2: (aninhado)

```
let primeiroSemestre = false,  
    segundoSemestre = false,  
    estagio = primeiroSemestre ? "Estágio não liberado" :  
segundoSemestre ? "Estágio não liberado" : "Estágio liberado";  
  
console.log( estagio );
```

*** Poderia colocar ()*

```
estagio = primeiroSemestre ? "Estágio não liberado" :  
(segundoSemestre ? "Estágio não liberado" : "Estágio  
liberado");
```


JavaScript – Operadores Especiais

O operador vírgula (,) avalia os operandos e retorna o valor do último (da esquerda para a direita). Este operador é utilizado primariamente dentro de um laço for para permitir que várias variáveis sejam atualizadas cada vez através do laço.

Exemplo 1:

```
function x() {  
    return 'A';  
}  
function y() {  
    return 'B';  
}  
function z() {  
    return 'C';  
}
```

let x = (x(), y(), z()); → vai retornar somente C

JavaScript – Operadores Especiais

Exemplo 2:

Por exemplo, se *a* é uma matriz bidimensional (9,9), o código a seguir utiliza o operador vírgula para incrementar duas variáveis de uma só vez.

```
for (var i=0, j=9; i<= 9; i++, j--) {  
    document.writeln "["+i+"["+j+"]" );  
}
```

*O código imprime os valores diagonais da matriz a: elementos [0][9] [1][8]
[2][7] [3][6] [4][5] [5][4] [6][3] [7][2] [8][1] [9][0]*

**** Ver UsandoOperadoresEspeciais1.html**

JavaScript – Operadores Especiais

O operador delete apaga um objeto, uma propriedade de um objeto ou um elemento no índice especificado de uma matriz.

```
delete nomeObjeto;
```

```
delete nomeObjeto.propriedade;
```

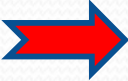
```
delete nomeObjeto[“propriedade”];
```

```
delete nomeMatriz[indice];
```


JavaScript – Operadores Especiais

O operador `in` retorna verdadeiro se a propriedade especificada estiver no objeto especificado. A sintaxe é: `nomePropriedadeOuNumero in nomeObjeto`

```
const nomes = new Array("Ana", "Laura", "Clara", "Isabela", "Isadora");
```



```
if (o in nomes) {  
    alert("existe a posicao o");  
}  
else {  
    alert("nao existe a posicao o");  
}
```



```
if ("length" in nomes) {  
    alert("existe a propriedade length em nomes (vem de array)");  
}
```


JavaScript – Operadores Especiais

O operador `new` serve para criar uma instância de um tipo de objeto definido pelo usuário ou de um dos tipos de objetos predefinidos:

Array, Boolean, Date, Function, Image, Number, Object, RegExp, String.

```
let data = new Date(2007, 06, 26);
```

```
function Carro() {}  
carro1 = new Carro()
```


DDAOVO

A utilização de expressões regulares, na maioria das linguagens de programação, representa um recurso de grande importância para a manipulação de strings. Em JavaScript, no entanto, devido às características do front-end das aplicações, onde é comum a entrada de dados pelo usuário, as RegEx passam a ter uma função ainda mais relevante, por estarem diretamente relacionadas à consistência das informações, por exemplo testar um texto de entrada cpf, telefone, nome, etc.

Expressões regulares são padrões utilizados para selecionar combinações de caracteres em uma string. Em JavaScript, expressões regulares também são objetos. Elas podem ser utilizadas com os métodos `exec` e `test` do objeto `RegExp`, e com os métodos `match`, `replace`, `search`, e `split` do objeto `String`.

```
let re = new RegExp("ab+c");  
function checkpostal(){  
    var re5digit=/^\d{5}$/ //regular expression defining a 5 digit number  
    if (document.myform.myinput.value.search(re5digit)=-1) //if match failed  
        alert("Please enter a valid 5 digit number inside form")  
}
```

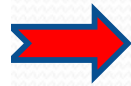
DENILCE DE ALMEIDA OLIVEIRA VEL; 2025-03-26T14:31:35.756

JavaScript – Operadores Especiais

O operador `instanceof` retorna verdadeiro se o objeto especificado for do tipo de objeto especificado. A sintaxe é:

`nomeObjeto instanceof tipoObjeto`

```
let data = new Date(2007, 06, 26);
```



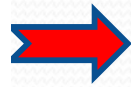
```
if (data instanceof Date) {  
    alert("data é instância");  
} else {  
    alert("data não é instância")  
}
```

```
let st1 = "String1"; /* aqui foi criada como literal e não com o Construtor */  
let st2 = new String("String2");
```

```
if (st1 instanceof String) {  
    alert("st1 é instancia");  
}
```

**** Ver**

[UsandoOperadoresEspeciais4.html](#)



```
if (st2 instanceof String) {  
    alert("st2 é instancia");  
}
```


JavaScript – Operadores Especiais

O operador `this` é utilizado para se referir ao objeto atual. Em geral, o `this` se refere ao objeto chamado em um método. Sintaxe:

`this["nomePropriedade"]`
`this.nomePropriedade`

```
function myFunc() {  
    alert(this);  
}
```

```
myFunc();
```

→ vai imprimir [object window]

```
let obj1 = {  
    teste : function ()  
    {  
        return (this);  
    }  
}
```

```
alert (obj1.teste()); → vai  
imprimir [object object]
```

** Ver

UsandoOperadoresEspeciais5.html

JavaScript – Operadores Especiais

O operador `typeof` serve para saber o tipo de dados ou identidade de alguma coisa.

```
let anObj = {job: "Sou um objeto!"};
```

```
let aNumber = 42;
```

```
let aString = "Sou uma string!";
```

```
alert(typeof anObj); // irá imprimir "object"
```

```
alert(typeof aNumber); // irá imprimir "number"
```

```
alert(typeof aString); // irá imprimir "string"
```

****** Ver `UsandoOperadoresEspeciais2.html`

JavaScript – Operadores Especiais

O operador void é utilizado das seguintes formas:
void (expression) ou void expression

O operador void especifica que uma expressão deve ser avaliada sem retorno de valor.

Nesse exemplo evita que o navegador siga o link:

```
<a href="javascript:void(o);" onclick="minhaFuncao()">Clique aqui</a>  
<script>  
function minhaFuncao() {  
    alert("Você clicou no link!");  
}
```

** Ver UsandoOperadoresEspeciais3.html

Slide 85

DDAOVO

Neste exemplo, quando você clica no link com o texto "Clique aqui", a função `minhaFuncao()` é chamada. O href do link contém `javascript:void(0);`, onde `void(0)` é usado para evitar que o navegador siga o link. Em vez disso, ele executa a função `minhaFuncao()` e exibe um alerta.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2023-09-21T00:58:25.572

JavaScript – Operadores Especiais

O operador de coalescência nula (??) é um operador lógico que retorna o seu operando do lado direito quando o seu operando do lado esquerdo é null ou undefined. Caso contrário, ele retorna o seu operando do lado esquerdo.

```
let texto = null ?? 'padrão 1';  
alert(texto); // retorna padrão 1
```

```
let numero = 0 ?? 55;  
alert(numero); // retorna 0
```

```
let x;  
let = undefined ?? 99;  
alert(x); // retorna 99
```

**** Ver UsandoOperadoresEspeciais6.html**

```
let texto2=null;  
alert(texto2 = null??'padrão2'); // retorna padrão2
```


JavaScript – Operadores Especiais

Exemplo: void e coalescência nula

```
function funcao1()  
{  
    return void 0;  
}
```

```
let retorno = funcao1() ?? "é null ou  
undefined";  
alert(retorno);
```


JavaScript - Operadores

A propriedade global **NaN** é um valor especial que significa *Not-A-Number* (não é um número). Usado para indicar quando uma operação tentou retornar um número e falhou, *exemplos: dividir número zero por zero, `Math.sqrt(-1)` etc.* Nos navegadores modernos, o NaN é uma propriedade somente leitura e não configurável. Mesmo quando não for este o caso, evite sobrescrevê-lo.

Situações:

```
NaN === NaN; // falso  
Number.NaN === NaN; // falso  
isNaN(NaN); // verdadeiro  
isNaN(Number.NaN); // verdadeiro
```

Para detectar a ocorrência de um 'NaN' (Not a number) é mais comum usar a função `Number.isNaN(variavel)`.

JavaScript – Exemplo: UsandoNaN.html(1)

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <meta charset="UTF-8">
  <title>Teste NaN</title>

<script>
  let numero = 0 / 0;
  if (Number.isNaN(numero)) {
    alert(" teste1 - é não número - verdadeiro");
  } else {
    alert("teste1 - não é não número - falso");
  }

  /** situações
  NaN === NaN; // falso
  Number.NaN === NaN; // falso
  isNaN(NaN); // verdadeiro
  isNaN(Number.NaN); // verdadeiro
  **/
```

Curiosidade:

Digite no console:

"b"||"a"||"a"||"a"

O que acontece??

JavaScript – Exemplo: UsandoNaN.html(2)

```
if (numero === numero) {  
    alert("teste2 - verdadeiro");  
} else {  
    alert("teste2 - falso");  
}  
  
if (numero.NaN === numero) {  
    alert("teste3 - verdadeiro");  
} else {  
    alert("teste3 - falso");  
}  
  
if (isNaN(numero)) {  
    alert("teste4 - verdadeiro");  
} else {  
    alert("teste4 - falso");  
}
```


JavaScript – Exemplo: UsandoNaN.html(3)

```
    if (isNaN(Number.numero)) {  
        alert("teste5 - verdadeiro");  
    } else {  
        alert("teste5 - falso");  
    }  
    </script>  
</head>  
  
<body>  
  
</body>  
  
</html>
```


EXERCÍCIO

Supondo o seguinte código, informe o que será impresso em **cada** console?

```
<script>  
  let j;  
  let i=0;  
  let y=i/0;  
  let z= 2/i;  
  let w=null;  
  console.log(j); →  
  console.log (y); →  
  console.log (z); →  
  console.log (w); →  
</script>
```



| Comparações de Igualdade | | | | | |
|--------------------------|-------------------|-------|-------|-----------|--|
| X | Y | == | === | Object.is | |
| undefined | undefined | true | true | True | |
| null | Null | true | true | True | |
| true | true | true | true | True | |
| false | false | true | true | True | |
| "foo" | "foo" | true | true | True | |
| { foo: "bar" } | X | true | true | True | |
| 0 | 0 | true | true | True | |
| 0 | false | true | false | False | |
| "" | false | true | false | False | |
| "" | 0 | true | false | False | |
| "0" | 0 | true | false | False | |
| "17" | 17 | true | false | False | |
| new String("foo") | "foo" | true | false | False | |
| null | undefined | true | false | False | |
| null | false | false | false | False | |
| undefined | false | false | false | False | |
| { foo: "bar" } | { foo: "bar" } | false | false | False | |
| new String("foo") | new String("foo") | false | false | False | |
| 0 | null | false | false | False | |
| 0 | NaN | false | false | False | |
| "foo" | NaN | false | false | False | |
| NaN | NaN | false | false | True | |
| +0 | -0 | True | True | False | |

Slide 93

DDAOV0

O `object.is` é parecido com o `===`, isto é não faz coerção de tipo (conversão automática). Exceções para NaN e +0 e 0

DENILCE DE ALMEIDA OLIVEIRA VEL; 2023-04-16T23:36:36.044

DDAOV0 0

O método `Object.is()` realiza uma comparação de identidade estrita entre dois valores, sem fazer coerção de tipo. Portanto, quando você compara +0 com -0 usando `Object.is()`, o método retorna `false`, porque +0 e -0 são representações diferentes de zero em JavaScript.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2023-04-16T23:38:58.230

DDAOV0 1

identidade estrita (strict equality) é um tipo de comparação em JavaScript que verifica se dois valores são idênticos, tanto em valor quanto em tipo de dado.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2023-04-16T23:41:00.214

Exercícios

1) Informe se o identificador está sendo criado de maneira literal (L) ou como objeto (O):

- () `let x = new Number(50);`
- () `let obj1 = {};`
- () `let obj2 = new Object();`
- () `let nome = "José";`
- () `let nome1 = new String("José");`
- () `let notas = ["do", "ré", "mi", "fá", "sol", "lá", "si"];`
- () `let frutas = new Array("laranja", "banana", "maçã");`



2) Informe o resultado dos consoles:

```
let numero1=33;  
let numero2="33";  
let numero3 = new String("33");  
let obj1 = {nome: "Maria"};  
let obj2 = obj1;  
let obj3 = {nome: "Maria"};  
let nome1 = new String("Maria");  
let nome2 = "Maria";  
let numero5 = null;  
let numero6;  
console.log(numero1===numero2);  
console.log(numero2===numero3);  
console.log(numero2===numero3);  
console.log(obj1==obj2);  
console.log(obj1==obj3);  
console.log(nome1===nome2);  
console.log(numero5==numero6);
```



DDAOVO

False, True,False,True,False,False,True
DENILCE DE ALMEIDA OLIVEIRA VEL; 2024-04-17T13:44:12.704

JavaScript – Precedência Operadores

```
if (("Jon".length * 2) / (2+1) === (2))  
{  
    alert("Ok é verdadeiro!");  
}  
else  
{  
    alert("É falso!");  
}
```



```
if( 4 % 2 === 0 ) {  
    alert("O primeiro número é par");  
} else {  
    alert("O primeiro número é impar");  
}
```


JavaScript – Conversões de Números

Existem três funções para conversão de valores não numéricos para números: `Number()`, `parseInt()` e `parseFloat()`

Number()

- ✓ Usada em qualquer tipo de dado;
- ✓ Valores **booleanos** retornam **1** para **true** e **0** para **false**;
- ✓ **Números** retornam o próprio **valor** ;
- ✓ **null** retorna **0** ;
- ✓ **undefined** retorna **NaN** ;
- ✓ Se a string contém apenas números, ela é sempre convertida para um número decimal (zeros à esquerda são ignorados) ;
- ✓ Se a string contém um formato de ponto flutuante válido, como em “1.1”, ela é convertida no valor numérico apropriado;
- ✓ Se a string contém um formato **hexadecimal** válido, como em “0xf”, ela é convertida para o **número inteiro** correspondente ;
- ✓ Uma **string vazia** (ou com espaços em branco) é convertida para **0** ;
- ✓ Qualquer outra string é convertida para **NaN** .

JavaScript – Conversões de Números

Number()

Exemplos:

```
let num0 = Number(" ") //0
let num1 = Number("Boa Noite!") //NaN
let num2 = Number(""); //0
let num3 = Number("000011"); //11
let num4 = Number(true); //1
let num5 = Number("1.125"); //1.125
let num6 = Number(5.78657); //5.78657
let num7 = Number("00.5"); //0.5
let num8 = Number("0b1010");//10
```


JavaScript – Conversões de Números

parseInt()

- ✓ **Usado para converter uma string;**
- ✓ Ignora espaços iniciais, até que o primeiro caractere diferente de espaço seja encontrado, se não for o sinal de mais, menos ou um número, retornará NaN;
- ✓ **String vazia (ou com espaços em branco)** retorna NaN ;
- ✓ Quando encontra um caractere válido, a conversão continua até o fim da string ou até que um caractere não numérico seja encontrado;
- ✓ Reconhece formatos decimal, octal e hexadecimal.

Exemplos:

```
let num10 = parseInt("1234 feijaonoprato"); //1234
let num20 = parseInt(""); //NaN
let num30 = parseInt("oxA"); //10 - hexadecimal
let num40 = parseInt("546.5"); //546
let num50 = parseInt("70"); //70 - decimal
let num60 = parseInt("oxf"); //15 - hexadecimal
let num70= parseInt("-50");// -50
let num80= parseInt(" 1 6 0");//1
let num90 = parseInt("546.8"); //546
```


JavaScript – Conversões de Números

parseInt()

Se for passado um radical como segundo argumento a conversão pode mudar:

•Exemplos:

```
let num1 = parseInt("10", 2); // 10 binário → 2 decimal
```

```
let num2 = parseInt("10", 8); // 10 octal → 8 decimal
```

```
let num3 = parseInt("10", 10); // 10 decimal → 10
```

decimal

```
let num4 = parseInt("10", 16); // 10 hexadecimal → 16
```

decimal

JavaScript – Conversões de Números

parseFloat()

- ✓ Similar a `parseInt()`, também usado para converter uma string, mas converte para um número de ponto flutuante;
- ✓ Um primeiro ponto decimal é válido, mas um segundo ponto, caso seja encontrado, é ignorado juntamente com o resto da string ;
- ✓ Zeros iniciais são sempre ignorados.

•Exemplos:

`let num100 = parseFloat("1234 feijaonoprato"); //1234 - inteiro`

`let num200 = parseFloat("oxA"); //→ o não converte`

`let num300 = parseFloat("22.5"); //22.5`

`let num400 = parseFloat("22.34.5"); //22.34`

`let num500 = parseFloat("0908.5"); //908.5`

`let num600 = parseFloat("3.125e7"); //31250000`

`let num700 = parseFloat("22.578"); //22.578`

`let num800 = parseFloat("22.578").toFixed(2); //22.58`

`let num900 = parseFloat("22.578").toPrecision(2); //23`

`let num1000 = parseFloat("22.578").toPrecision(3); //22.6`

Converte arredondando
com 2 casas decimais

Converte arredondando
para nº de dígitos

JavaScript – Conversões para String

toString()

O método toString() está disponível em valores que são **números, booleanos, objetos e strings**

```
let idade = 30;  
let idadeAsString = idade.toString(); // “30”  
let estudante = true;  
let estudanteAsString = estudante.toString(); // “true”  
let nada;  
let nadaAsString = nada.toString(); // não imprime nada
```

****** Se um valor é null ou undefined, este método não está disponível

JavaScript – Conversões para String

String()

Essa função sempre retorna uma string, independentemente do tipo do valor .

- ✓ Se o valor possui o método toString(), ele é chamado e o resultado é retornado;
- ✓ Se o valor é null, “null” é retornado;
- ✓ Se o valor é undefined, “undefined” é retornado;
- ✓ Também é possível converter um valor para uma string adicionando uma string vazia (“”) ao valor usando o operador mais

JavaScript – Conversões para String

Exemplos:

```
let value1 = 20;  
let value2 = true;  
let value3 = null;  
let value4;  
let value5 = 22.5;
```

```
alert(String(value1)); //"20"  
alert(String(value2)); //"true"  
alert(String(value3)); //"null"  
alert(String(value4)); //"undefined"  
alert(String(value5)); //"22.5"  
alert(value5 + ""); //"22.5"
```


JavaScript – Método eval

A função **eval()** avalia código JavaScript representado como uma string.

Exemplos:

```
let x = 10;
```

```
let y = 20;
```

```
let a = eval("x * y"); //200
```

```
let b = eval("2 + 2"); //4
```

```
let c = eval("x + 17"); //27
```

```
let sTeste="10";
```

```
let nTeste = eval(sTeste);
```

```
alert(typeof(nTeste)); → number
```

DDAOVO

O uso do `eval()` pode ser útil em situações em que o código precisa ser gerado dinamicamente, como na criação de funções ou expressões matemáticas complexas em tempo de execução.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2023-04-16T23:42:15.940

JavaScript - Conversão Boolean

```
console.log(Boolean(5)); // true  
console.log(Boolean(0)); // false  
console.log(" "); // false
```

O método map do array permite transformar um array em outro array, aplicando uma função ao array original.

```
const strings = ["a", "b", "c"];  
const upperCaseStrings = strings.map(str => str.toUpperCase());  
console.log(upperCaseStrings); // ["A", "B", "C"]
```

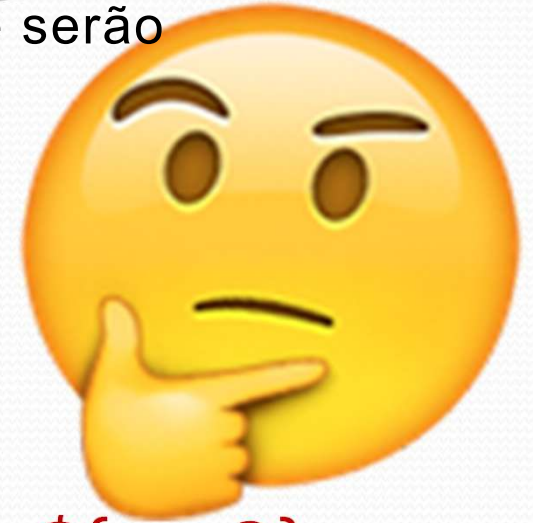
// convertendo para o boolean

```
const a= ["" , {}, {nome:"Zé"}, 0, 5, null, undefined, 'string'];  
console.log(a.map(Boolean));  
// false, true, true, false, true, false, false, true
```


Exercícios

1) Verificar quais são os resultados das conversões, que serão impressos nos alerts:

```
let num1 = parseInt("12 abc");  
let num2 = parseInt("");  
let num3 = parseFloat("12.52");  
let num4 = parseFloat("12.52.5");  
alert(`num1:${num1} num2:${num2} num3:${num3}  
num4:${num4}`);  
let num5;  
let num7 = String(num5);  
alert(`num7:${num7}`);  
let num8 =25;  
let num9 = eval("num8 + 10");  
alert(`num9:${num9}`);  
let num6 = num5.toString();  
alert(`num6:${num6}`);
```



JavaScript – Classe Math - Propriedades

A classe **Math** permite **operações matemáticas**. Não é necessário **instanciar** a classe para utilizar, pois os **atributos** e **métodos** são **variáveis de classe**, portanto basta utilizar o nome da classe para **acessá-los**.

Algumas **propriedades**:

E: retorna o valor de **Euler**, a base dos **logaritmos neperianos**. $e = 2,718.....$

PI: retorna o valor de **PI** conhecido como número para cálculos com círculos.

SQRT2: retorna o valor da **raiz quadrada** de 2;

SQRT1_2: retorna o valor da raiz quadrada de **0.5** ou $1/2$;

LN2: retorna o valor do Logaritmo neperiano de 2 ($\ln 2$);

LN10: retorna o valor do Logaritmo neperiano de 10 ($\ln 10$);

LOG2E: retorna o valor do Logaritmo de E (Math.E) na **base 2**;

LOG10E: retorna o **valor** do Logaritmo de E na base 10.

Alguns métodos:

abs(): valor absoluto de um número.

acos(): arco cosseno de um número em radianos.

atan(): o arco tangente de um número.

ceil() / floor(): o inteiro igual ou imediatamente seguinte de um número (ou inferior -> floor).

cos(): o co-seno de um número.

exp(): o resultado de elevar o número E por um número.

log(): o logaritmo neperiano de um número.

max(): o maior entre 2 números.

min(): retorna o menor entre 2 números.

pow(): primeiro número elevado ao segundo número.

random(): retorna um número aleatório entre 0 e 1.

round(): Arredonda ao inteiro mais próximo.

sin(): seno de um número com um ângulo em radianos.

sqrt(): raiz quadrada de um número.

tan(): tangente de um número em radianos.

5 Criando um jogo

"Pedra, papel ou tesoura" é um jogo clássico para 2 pessoas. Cada jogador escolhe pedra, papel ou tesoura. Neste exercício o jogador irá jogar com o computador. Os resultados possíveis são:

Empate.

Pedra quebra tesoura.

Tesoura corta papel.

Papel cobre a pedra.

Fases:

a. O usuário faz uma escolha

b. O computador faz uma escolha (método random – ponto flutuante aleatórios $[0, 1)$, 0 (inclusivo) até, mas não incluindo, 1 (exclusivo)). Sugestão divida 0.99 em 3 partes.

c. A partir das duas escolhas determinar o vencedor

→ Disponibilizar no GitHub → seuusuario\PWEB\Atividade7



DICAS

Pode usar inputs, botões etc

****** se carregar o js no head e executar direto, cuidado para não se referir a elementos html que ainda não foram criados. É melhor colocar em funções dentro do js. Se os elementos estiverem dentro do form ele realiza o submit (recarrega página)

Criando uma pesquisa

Durante o lançamento de um filme, 45 pessoas assistiram. Na saída foi realizada uma pesquisa informando a idade, sexo (feminino, masculino e outros) e a opinião, onde: ótimo=4, bom=3, regular=2, péssimo=1.

Fazer uma aplicação utilizando JavaScript que receba os dados (idade, sexo e opinião) e retornar:

- a média da idade das pessoas que responderam ao questionário;
- a idade da pessoa mais velha;
- a idade da pessoa mais nova;
- a quantidade de pessoas que responderam péssimo;
- a porcentagem de pessoas que responderam ótimo e bom;
- o número de mulheres, homens e outros que responderam ao questionário.

→ Disponibilizar no GitHub → seuusuario\PWEB\Atividade8

Lembrando que:

$$\text{Média} = (\text{Avaliação Teórica} * 35\% + \text{Site} * 35\% + \text{Média Atividades extras} * 30\%)$$

Vamos pensar no nosso projeto?



.htm - vem do do MS-DOS depois Windows (extensão três letras, por exemplo: .doc, .exe, .txt, etc.)_

.html - vem do UNIX que suportava mais de três caracteres.

Para evitar problemas, foi feita a junção e, para efeitos de compatibilidade, foram mantidas as duas extensões. **Não há diferença entre htm e html**, ou seja, o conteúdo das páginas são os mesmos. Se houver em um mesmo site dois documentos com o mesmo nome, um arquivo chamado index.html e outro, index.htm, o servidor irá sempre apresentar o index.html.

.xhtml - mesmo HTML, porém, ele é mais restritivo quando a forma de usar as tags. O termo XHTML é na verdade a junção da linguagem HTML com as especificações do XML, daí o X.

dhtml - **Dynamic HTML**, ou **DHTML**, usado para desenvolver páginas Web interativas que podem mudar sem ser recarregada ou abrir outro link. Frequentemente contido dentro de um html regular.

Por onde começar???

ROADMAP.SH – MAPA APRENDIZAGEM (1)

Roadmap.sh é um site gratuito que oferece guias de estudo para aprender programação e tecnologia. O site é uma iniciativa comunitária que disponibiliza "mapas de carreira" (roadmaps) para diversas áreas da tecnologia. <https://roadmap.sh/>

- Roadmaps detalhados que indicam o que aprender e em que ordem
- Recomendações sobre como melhorar a qualidade do código, segurança e desempenho
- Recursos para colocar em prática o que você aprendeu
- Uma comunidade de desenvolvedores que trocam experiências e ajudam uns aos outros
- Tópicos que vão desde o básico para iniciantes até caminhos avançados para profissionais experientes

RODMAP.SH – MAPA APRENDIZAGEM (2)

Quais áreas são contempladas?

- Desenvolvimento web
- Desenvolvimento mobile
- DevOps
- Ciência de dados
- Computação
- Design e arquitetura
- Análise de dados
- Segurança cibernética
- UX design
- Desenvolvimento de jogos

Referências

CODEACADEMY. Cursos Gratuitos. <https://www.codecademy.com/pt> Acesso em: Jan. 2025.

CROCKFORD, Douglas. <http://crockford.com/javascript/> Acesso: Mai.2021.

CSS. <http://del.icio.us/carlosbazilio/{css+html}> Acesso em: Jan.2015.

JS. Livro de JavaScript. <https://github.com/getify/You-Dont-Know-JS> Acesso em: Jan.2021.

JS1. <http://www.significados.com.br/javascript/> Acesso em: Jan.2015

JS2. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/JavaScript_Vis%C3%A3o_Geral Acesso em: Jan.2025

SILVA. Maurício Samy . JavaScript Guia do Programador. Editora Novatec.

JSOBJETOS. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Trabalhando_com_Objeto Acesso em: Jan.2025.

MOZILLA. JavaScript.. <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript> Acesso em: Set.2024.

OPERADORES. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#Assignment_operators Acesso em: Jan.2025.

PEREIRA. Fábio M. Pereira. JavaScript Básico. DESENVOLVIMENTO DE SISTEMAS WEB – 2014.1 UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA CURSO DE CIÊNCIA DA COMPUTAÇÃO. 2014.

PRECEDENCIA. [https://msdn.microsoft.com/pt-br/library/z3ks45k7\(v=vs.94\).aspx](https://msdn.microsoft.com/pt-br/library/z3ks45k7(v=vs.94).aspx) Acesso em: Jan.2025.