

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 e ES2020.
- 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)* - controle sobre a estrutura da página e conteúdo de forma hierárquica;
- ✓ O *Browser Object Model (BOM)* (objeto window).



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

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++;
- ✓ Tem fortes capacidades de programação orientada a objetos, apesar das discussões devido às diferenças da orientação a objetos no JavaScript em comparação com outras linguagens (**versões mais novas com ECMAScript já são mais completas nesse aspecto**);
- ✓ Comandos são Case Sensitive.

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

JavaScript X Java

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

O processamento síncrono, são todos os processamentos que 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%A1s-da-programa%C3%A7%C3%A3o-funcional-do-java-8-bd8cofi72d45>

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

JavaScript X Java

Programação Funcional

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 ]
```

<https://www.alura.com.br/artigos/programacao-funcional-o-que-e>

Princípio da melhoria progressiva

- ✓ Usar a linguagem com o propósito único de incrementar a usabilidade da página;
- ✓ Segundo esse princípio uma página Web deve ser feita em 3 etapas:
 - 1) Estruturar os conteúdos usando a linguagem HTML. Ao final dessa etapa, todos os conteúdos devem estar disponíveis para qualquer visitante que esteja utilizando qualquer dispositivo de usuário;

Princípio da melhoria progressiva

- 2) Incrementar a apresentação da página com o uso das CSS. Essa etapa visa a melhorar a experiência dos usuários aptos a visualizar folhas de estilo;
- 3) Introduzir JavaScript com a finalidade de acrescentar interatividade à página, melhorando ainda mais a experiência do usuário.

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, poderá ter problemas, pois ele deixará todos os comandos em uma linha só. Ou em casos mais específicos como o exemplo que será mostrado, poderá não chegar ao resultado desejado. Então, é considerado uma boa prática colocar o ";" para evitar erros de omissão e até melhorar a performance em alguns casos.

** 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. Para strings e expressões regulares, são considerados. 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 = 2 7; // espaço entre os algarismos de um número não é permitido`

`document.write("<p> Eu sou`

`uma string</p>") // quebra de linha em uma string`

JavaScript – Espaços em branco e quebras linha

✓ A sintaxe a seguir é válida:

```
document.write("<p> Eu sou \  
uma string</p>") // uso de \ para quebrar linha em string
```

✓ A sintaxe a seguir causa um erro no script:

```
document.write \  
("<p> Eu sou uma string</p>") // uso de \ para quebrar  
linha fora de string
```

✓ A sintaxe a seguir é válida:

```
document.write  
("<p> Eu sou uma string</p>") // quebra de linha entre  
nome de funções
```


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 - 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 a sua primeira página utilizando JavaScript para imprimir Alô Mundo!

Padrões no JavaScript

- Ver <https://gist.github.com/vinicius-stutz/1b37cb84b6240efe6ab8137660a15640#:~:text=Segundo%20Douglas%20Crockford%2C%20todos%20os,como%20o%20C%23%2C%20por%20exemplo.>
- <https://www.crockford.com/code.html>

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`

✓ **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

Fonte: <https://www.alura.com.br/artigos/entenda-diferenca-entre-var-let-e-const-no-javascript>

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
```

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

```
nome = "Matheus",
```

```
idade = 30,
```

```
turma = "A";
```

JavaScript – Regras de escopo para variáveis

- Se você declarar uma variável com a palavra-chave `var` em uma função ou em um bloco de código, seu uso é local àquela função.
- Se você usar uma variável sem declará-la com a palavra-chave `var`, e existir uma variável global com o mesmo nome, a variável local é assumida como sendo a global já existente.
- Se você declarar uma variável localmente com uma palavra-chave `var`, mas não a inicializar (i.e., atribuir um valor a ela), ela será local e ficará acessível, mas não definida.
- Se você declarar uma variável localmente sem a palavra-chave `var`, ou se declará-la explicitamente como global mas não a inicializar, ela será acessível globalmente, mas também não estará definida.

JavaScript – Variáveis – hoisting (1)

Em JavaScript é possível utilizar a variável e declará-la depois, sem obter uma exceção. Este conceito é conhecido como hoisting ou “elevação” (não é muito comum). No entanto, as variáveis que são "hoisted" retornarão um valor undefined.

```
/**
```

```
 * Exemplo
```

```
 */
```

```
console.log(x === undefined); // testando se o x é  
undefined, vai voltar true
```

```
var x = 3;
```

JavaScript – Variáveis – hoisting (2)

```
<script>
  function mostratexto()
  {
    texto1 = "ESTE É O TEXTO 1";

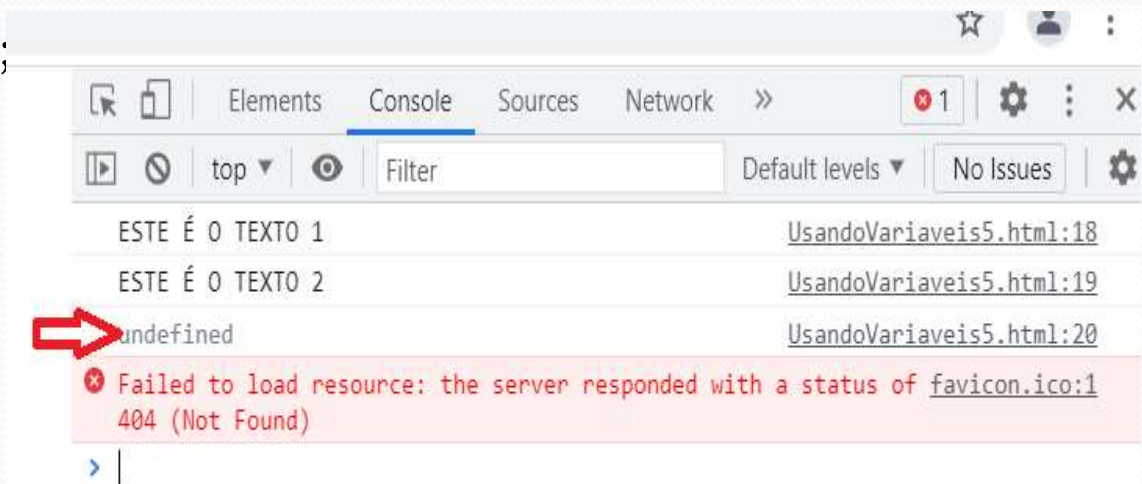
    if (true)
      var texto2 = "ESTE É O TEXTO 2";

    console.log(texto1);
    console.log(texto2);
    console.log(mensagem1);

    var mensagem1;
  }

  mostratexto();

</script>
```



JavaScript – Exemplo: Usando Variaveis1.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; /* assume como a global já existente */
```

```
    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);
```

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

```
<body>  
</body>
```

```
</html>
```

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 aspas simples, deve terminar com aspas simples, da mesma forma para as aspas duplas.

JavaScript - Criando Variáveis

Tipos e objetos nativos ECMAScript

function

Tipo de objeto que representa funções, métodos e construtores

object

Tipo de dados nativo que representa coleções de propriedades contendo valores de tipos primitivos, function ou object

Objetos nativos embutidos

Object

Array

Global

Boolean

Number

String

Function

Date

Math

Tipos de dados primitivos (que representam valores)

undefined

representa valores ainda não definidos

undefined

null

representa o valor nulo

null

boolean

representa valores booleanos

true
false

number

representa números de ponto-flutuante IEEE 754 com precisão de 15 casas decimais (64 bits)

Min: $\pm 4.94065 \text{ e-}324$

Max: $\pm 1.79769 \text{ e+}308$

NaN
Infinity
-Infinity

string

representa cadeias ordenadas (e indexáveis) de caracteres Unicode.

"\u0000 - \uFFFF"

'\u0000 - \uFFFF'

''

""

"abcde012+\$_@..."

** 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, a saber:

Caractere	Descrição – Caractere Unicode hexadecimal
\b	Backspace – \u0008
\f	Form feed – \u000C
\n	Nova linha – \u000A
\r	Retorno do carro – \u000D
\v	Tabulação vertical – \u000B
\t	Tabulação horizontal – \u0009
\'	Apóstrofo ou aspas simples – \u0027
\"	Aspas duplas – \u0022
\\	Barra invertida – \u005C
\XXX	Caractere Latin-1 expresso por dígitos octais de 1 a 377
\xxx	Caractere Latin-1 expresso por dois dígitos hexadecimais de 00 a FF
\uXXXX	Caractere Unicode expresso por quatro dígitos hexadecimais

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.

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

A representação literal para NULL é a string 'null' sem os delimitadores. quando referenciado por uma função ou comando de tela. NULL é uma palavra reservada.

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>
```

JavaScript - Criando Variáveis

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

```
var alunoBolsista = True; //boolean
```

```
var intNum = 55; // inteiro
```

```
var octalNum1 = 070; //octal para 56 (precedido de 0, usa 0 .. 7)
```

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

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

→ Nos 3 casos acima será impresso valor 10

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

```
var floatNum1 = 1.1;
```

```
var floatNum2 = 0.1;
```

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

** *ver UsandoVariaveis3.html*

JavaScript - Variáveis String

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
  <title>Testando IF</title>
  <script>
    var 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. `10/0`

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:

```
var soma1 = "3" + 4 + 5;    -> 345  
var soma2 = 3 + 4 + "5";   -> 75 (concatena)  
var soma3 = 3 + "4" + 5;   -> 345
```

```
var numero1 = 23  
var numero2 = "42"  
var soma = numero1 + numero2 -> 2342 (concatena)
```

JavaScript – Operadores Numéricos

O que será impresso no código abaixo?

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

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



JavaScript – Operadores Numéricos

O que será impresso no código abaixo?



```
var 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???

JavaScript – Operadores Unários

✓ Incremento de Prefixo e Sufixo (++)

Supondo:

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

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

JavaScript – Operadores Comparação

O que será impresso no código abaixo?

```
if ("3"===3){  
    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.

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

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

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

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

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 da operação?

$8 \wedge 2$



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!");
    var nome;

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

    alert("Seu nome é " + nome);

    var 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 externo.

Disponibilizar no GitHub →
seuusuario\PWEB\Atividade6

- 1) **Media.html** : Leia o nome e as três 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

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:

```
var fatec = "Sorocaba";  
var semestre = 4;
```

```
var 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:

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


JavaScript – Operadores Especiais

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

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]*

JavaScript – Operadores Especiais

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

`delete nomeObjeto;`

`delete nomeObjeto.propriedade;`

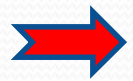
`delete nomeObjeto[indice];`

`delete propriedade; // válido apenas dentro de uma declaração with`

JavaScript – Operadores Especiais

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

```
var 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.

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

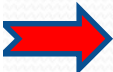
```
function Carro() {}  
carro1 = new Carro()
```


JavaScript – Operadores Especiais

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

nomeObjeto **instanceof** tipoObjeto


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



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

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

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



```
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]

JavaScript – Operadores Especiais

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

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

```
var aNumber = 42;
```

```
var 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)

void expression

O operador void especifica que uma expressão deve ser avaliada sem retorno de valor. Os parênteses em torno da expressão são opcionais, mas faz parte de um bom estilo utilizá-los.

Por ex. utilizar o operador void para especificar uma expressão como um link de hipertexto. A expressão é avaliada mas não é carregada no lugar do documento atual.

** Ver UsandoOperadoresEspeciais3.html

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, *por exemplo, dividir qualquer número por zero*. 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>
    var numero = o / o;
    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>
```


If (Object.is(x,y))

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	

Ver Exemplo: UsandoIgualdades.html

Adaptado de:
<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Igualdade>

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 é convertida para 0 ;
- ✓ Qualquer outra string é convertida para NaN .

JavaScript – Conversões de Números

Number()

Exemplos:

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

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

JavaScript – Conversões de Números

parseInt()

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

•Exemplos:

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

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

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

```
var 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:

```
var num100 = parseFloat("1234 feijaonoprato"); //1234 - inteiro
var num200 = parseFloat("0xA"); //10
var num300 = parseFloat("22.5"); //22.5
var num400 = parseFloat("22.34.5"); //22.34
var num500 = parseFloat("0908.5"); //908.5
var num600 = parseFloat("3.125e7"); //31250000
var num700 = parseFloat("22.578"); //22.578
var num800 = parseFloat("22.578").toFixed(2); //22.58
var num900 = parseFloat("22.578").toPrecision(2); //23
var 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

```
var idade = 30;  
var idadeAsString = idade.toString(); // "30"  
var estudante = true;  
var estudanteAsString = estudante.toString(); // "true"  
var nadaAsnull;  
var 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, independente 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:

```
var value1 = 20;
```

```
var value2 = true;
```

```
var value3 = null;
```

```
var value4;
```

```
var 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:

```
var x = 10;
```

```
var y = 20;
```

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

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

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

JavaScript – Instrução Condicional IF

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>
    var 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 – Classe Math - Propriedades

A classe **Math** permite **operações matemáticas**. Não é necessário **instanciar** a classe para utilizar, pois os **atributos** e **método** 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.

JavaScript – Classe Math - Métodos

Alguns métodos:

abs(): valor absoluto de um número.

acos(): arco co-seno de um número em radianos.

atan(): o arco tangente de um número.

ceil() / **floor()**: o 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.

Criando um jogo

"Pedra, papel ou tesoura" é um jogo clássico para 2 pessoas. Cada jogador escolhe pedra, papel ou tesoura. 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

Velocidade da luz

Utilizando JavaScript converta a velocidade da luz de metros / segundo para centímetro / nanossegundo.

Dados:

velocidade da luz = 299792458 metros / segundo

1 metro = 100 centímetros

1 nanossegundo = 1.0 / 1000000000 segundo

Velocidade da luz (resolução)

299792458 m/s 100 cm/m 0.000000001 s/ns

Ou seja $299792458 * 100 * 1 / 1000000000$

A resposta é 30 cm/ns.

Criando uma pesquisa

Durante o lançamento de um filme, 45 pessoas assistiram. Na saída foi realizada uma pesquisa informando a idade, sexo 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 e homens que responderam ao questionário.

→ Disponibilizar no GitHub → seuusuario\PWEB\Atividade8

Dica por onde começar

<http://programadorobjetivo.co/o-melhor-caminho-para-aprender-javascript-e-domina-lo/>

Livros

<https://woliveiras.com.br/posts/Livros-sobre-JavaScript-do-iniciante-ao-avancado-e-ES6/>

Cursos

<https://braziljs.org/blog/os-melhores-cursos-online-para-aprender-javascript/>

Referências

CAELUM. Apostilas Cursos Gratuitas <https://www.caelum.com.br/apostilas/>. Acesso em: Jan.2015.

CODEACADEMY. Cursos Gratuitos. <https://www.codecademy.com/pt> Acesso em: Jan. 2015.

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.2015

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.2015.

OPERADORES. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#Assignment_operators Acesso em: Jan.2015.

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.2015.