



A linguagem de programação JavaScript

Prof. Luiz Henrique de Angeli
 luizdeangeli@gmail.com
 luiz.angeli@unicesumar.edu.br

Quem sou?



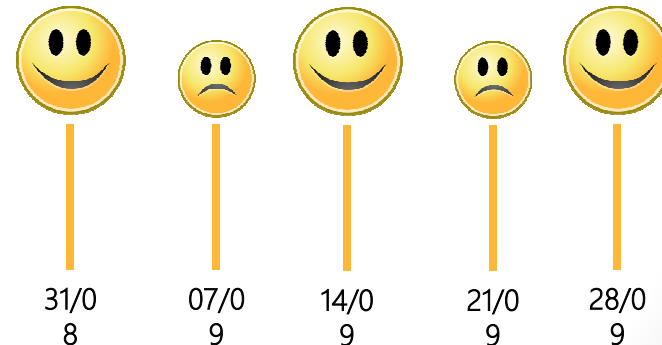
- **Luiz Henrique de Angeli**
- Graduado em Processamento de Dados
Unicesumar – Centro Universitário Cesumar (2006)
- Pós-Graduado em Desenvolvimento de Sistemas Orientado a Objetos Java
Unicesumar – Centro Universitário Cesumar (2010)
- Professor na Unicesumar desde 2007
Ministrando disciplinas na área de desenvolvimento: PHP, HTML, CSS, JS, DELPHI, SHELL SCRIPT, C, PASCAL
- Head de Desenvolvimento Ensino EAD
- Desenvolvimento para WEB desde 2005

Conteúdo da Disciplina



- | | |
|---------------------------|------------------------|
| • O que é | • Funções |
| • História | • DOM |
| • Como programar | • Objetos |
| • Funcionalidades | • Tratamento de Erros |
| • Caixa de Diálogo | • Expressões Regulares |
| • Erros e Depuração | • ECMAScript 6+ |
| • Tipos de dados | • Testes unitários |
| • Operadores | • TypeScript |
| • Variáveis | • JQuery |
| • Estrutura condicional | • Tópicos especiais |
| • Estrutura de repetições | |
| • Arrays | |

Nossas aulas....

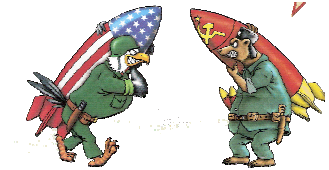


Um pouco da história da internet

é rapidinho.....

- Durante a guerra fria, a **Darpa** criou uma rede de dados chamada **Arpanet** para comunicar 4 computadores e criaram os protocolos de comunicação:

- NCP : Protocolo de Controle de Rede
- FTP: Transferência de arquivos
- DNS: Identificação de maquinas



- Objetivo da rede era proteger os dados em casos de ataques;
- Dinheiro do Governo dos EUA + Conhecimento dos Universitários

1969

- **Ray Tomlinson** criou um sistema para troca de mensagens online o **email**;

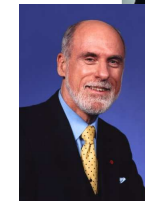


1972

- **Robert Kahn**: Modificiou o NCP e criou o TCP (Transfer Control Protocol) e criou o termo Internetting;

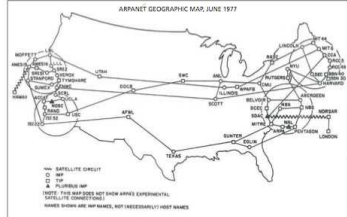


- **Vint Cerf** e criou o (Internetwork Protocol) que juntos criaram o TCP/IP



1973

- Governo dos EUA e os militares não quiseram mais tomar conta do projeto e foi criada a **Internet**



1977

- **Timothy John Berners-Lee** criou:
 - O protocolo **http** (hipertext transfer protocolo);
 - **A linguagem html**: HyperText Markup Language;
 - **WWW**: World Wide Web
 - Criou –se então a primeira versão do HTML



1990

- A NCSA criou Navegador Mosaic que ficou muito famoso na época;



1992

- Desenvolvedores do NCSA saíram do órgão e criaram a empresa Netscape e também criaram o navegador com o mesmo nome;



1994

- Foi criado os navegadores:

- IBM WebExplorer
- UDI WWW
- Internet Explorer 1.0

- Briga entre Microsoft e Netscape.

- Microsoft teve que retirar o IE do Windows



1995

- Netscape foi vendida para America Online
- Fundadores do Netscape criaram a Fundação Mozilla e o navegador Mozilla Firefox



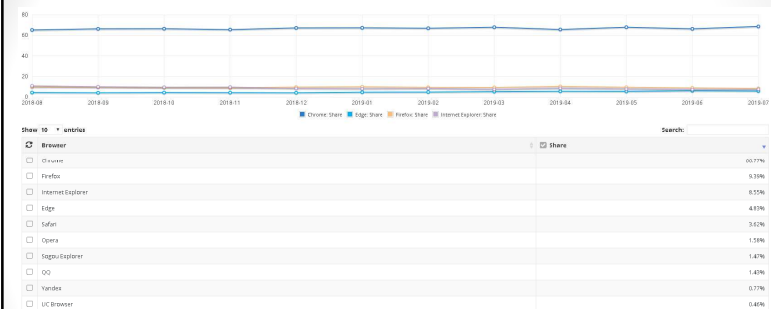
1999

- A Google resolveu entrar na briga e criou o seu próprio navegador;
- Em 2 anos de criação passou a ser o 3º navegador mais utilizado



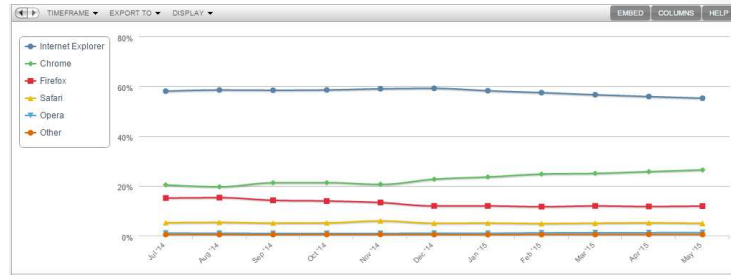
2008

- Como esta o uso dos navegadores em geral?



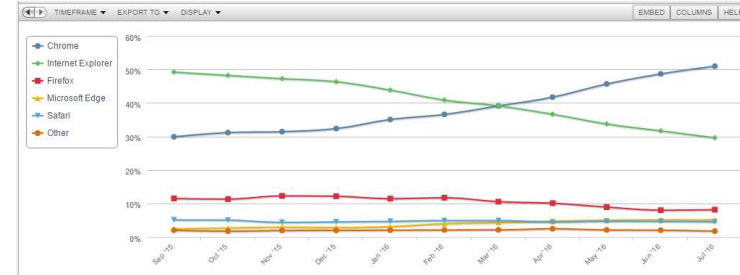
2019

- www.netmarketshare.com



2015

- www.netmarketshare.com



2016

W3C



- No mês de outubro de **1994**, foi criado o World Wide Web Consortium (W3C);
- O Consórcio World Wide Web (W3C) é um consórcio internacional no qual organizações filiadas, uma equipe em tempo integral e o público trabalham juntos para desenvolver padrões para a Web.
- Liderado pelo inventor da web **Tim Berners-Lee** e o CEO **Jeffrey Jaffe**, o W3C tem como missão **Conduzir a World Wide Web para que atinja todo seu potencial, desenvolvendo protocolos e diretrizes que garantam seu crescimento de longo prazo.**
- <http://www.w3.org>
- <http://www.w3c.br>

O WHATWG (The Web Hypertext Application Technology Working Group) é um grupo de desenvolvedores que não se conformavam com a velocidade que as coisas aconteciam no W3C. Portanto eles são um grupo separado do W3C.

Esse grupo foi fundado por membros da Mozilla, Apple e Opera por volta de 2004. Eles resolveram iniciar a escrita de uma nova recomendação para o HTML, já que o W3C iria abandonar o HTML e focar seus esforços em linguagens baseadas em XML.

É por isso que algum dia no seu passado você começou a fechar as tags BR com uma barra no final (
).

19

19

19

19

20

20

20

As Três Camadas da Web



- HTML: [Conteúdo]
 - Camada de base que qualquer visitante deve ser capaz de ver
 - Tudo que é necessário para entender o conteúdo de uma página web.
- CSS: [Apresentação]
 - Camada para apresentar uma melhor aparência para o site
- JAVASCRIPT: [Comportamento]
 - Camada de interatividade e comportamento dinâmico



As Três Camadas da Web

JAVASCRIPT
[Comportamento]

CSS
[Apresentação]

HTML
[Conteúdo]



Riqueza na
experiência
com o usuário

E o JavaScript???

O que é JavaScript



- É uma **linguagem de programação** interpretada;
- Esta presente nos principais navegadores;
- É atualmente a principal linguagem de programação *cliente/servidor* em navegadores web;
- É leve e expressivo;
- Curva de aprendizagem pequena;
- É uma linguagem com muitas contradições, contém muitos erros e pontas soltas;

Timeline do JavaScript

- A empresa **Nombas** criou um projeto cmm (c--), que seria uma linguagem mais simples, por questões de aprendizagem;
- Foi comprada em pouco tempo pela **Openwave**;
- Os desenvolvedores resolveram rebatizar a linguagem para **ScriptEase**

1992



- Em seguida o ScriptEase despertou um interesse em uma grande empresa da época, a **Netscape**;
- Assim a Netscape pegou o projeto do ScriptEase para utilizar em seu produto, o navegador Netscape;

1994

- O projeto foi parar na mão de **Brendan Eich** que rebatizou o projeto para **Mocha**;
- Posteriormente antes do lançamento a linguagem foi rebatizada para **LiveScript**
- O LiveScript foi lançado no Netscape 2.0



1995




- No mesmo ano do lançamento a **Netscape** estava em negociação com a **SUN** para desenvolver novas tecnologias para navegadores;
- A **SUN** estava acabando de fazer o grande o lançamento, a linguagem **JAVA**;
- E fizeram uma grande “jogada de marketing” mudando o nome da linguagem para **JavaScript**;

1995

JavaScript	Java
Orientada a objeto. Sem distinção entre tipos e objetos. A herança é feita através do protótipo e as propriedades e métodos podem ser adicionadas a qualquer objeto dinamicamente.	Baseada em classes. Objetos são divididos em classes e instâncias com toda a herança através da hierarquia da classe. Classes e instâncias não podem ter propriedades ou métodos adicionados dinamicamente.
Os tipos de dados das variáveis não precisam ser declarados (tipagem dinâmica)	Os tipos de dados das variáveis devem ser declarados (tipagem estática).
Não pode escrever automaticamente no disco rígido.	Pode escrever automaticamente no disco rígido.
Linguagem não compilada	Linguagem compilada

1995

- Microsoft resolveu criar uma linguagem para o seu navegador **Jscript**;

Microsoft 

1996

- A **Netscape** submeteu o JavaScript a **ECMA** internacional como candidato a padrão industria e o trabalho subsequente resultou na versão padronizada chamada **ECMAScript**;
- ECMA é uma associação dedicada a padronização de sistemas de informação;
- Surgiu o ECMA-262 – ECMAScript: uma versão padronizada do JavaScript. Foi uma jogada para ter o reconhecimento do padrão do mercado.

1997

EcmaScript (versões)

Edição	Publicação	Principais Aspectos
1	Junho, 1997	Primeira versão.
2	Junho, 1998	Pequenas modificações para manter a especificação alinhada com a ISO/IEC 16262.
3	Dezembro, 1999	Foram adicionadas expressões regulares, melhorias no tratamento de strings, definição de erros, tratamento de exceções com try/catch, formatação para output de valores números e outras melhorias.
4	Abandonada	A quarta versão propunha modificações grandes, como a inclusão de classes, modularização, generators e afins. Infelizmente, acabou sendo abandonada devido ao avanço lento e complicações políticas entre os membros da Ecma's Technical Committee 39 (TC39).

EcmaScript (versões)

Edição	Publicação	Principais Aspectos
5	Dezembro, 2009	Tornou-se mais claro muitas ambiguidades presentes na terceira edição da especificação. Além disso, adicionou algumas novas features, tais como: getters e setters, suporte a JSON e um mecanismo mais robusto de reflexão nas propriedades dos objetos.
5.1	Junho, 2011	Adaptações para que a especificação ECMA ficasse totalmente alinhada com o padrão internacional ISO/IEC 16262:2011.
6	Junho, 2015	Versão ES6
	2016	Versão ES2016
	2017	Versão ES2017
	2018	Versão ES2018
	2019	Versão ES2019

Visão Geral

- Não há como fazer funcionar um formulário HTML com o uso de elementos HTML.
 - A HTML limita-se a criar os rótulos e campos de um formulário para serem preenchidos pelo usuário e nada mais.

Visão Geral

- **O lado cliente do JavaScript** fornece objetos para controlar um navegador web e seu *Document Object Model* (DOM).
- Por exemplo, as extensões do lado do cliente permitem que uma aplicação coloque elementos em um formulário HTML e responda a eventos do usuário, como cliques do mouse, entrada de formulário e de navegação da página.

Visão Geral

- **O lado do servidor do JavaScript** fornece objetos relevantes à execução do JavaScript em um servidor.
- Por exemplo, as extensões do lado do servidor permitem que uma aplicação comunique-se com um banco de dados, garantindo a continuidade de informações de uma chamada para a outra da aplicação, ou executar manipulações de arquivos em um servidor.

Funcionalidades gerais da JavaScript

- Manipular conteúdo e apresentação
- Manipular o navegador
- Interagir com formulários
- Interagir com outras linguagens dinâmicas

Orientação a objetos

- A linguagem JavaScript suporta programação orientada a objetos – Object-Oriented Programming (OOP)
- É mais apropriado dizer que JavaScript é uma linguagem capaz de simular muitos dos fundamentos de OOP, embora não plenamente alinhada com todos os conceitos de orientação a objetos

Popularização do Javascript

- O Javascript começou a ser visto com outros olhos após o crescimento da utilização do Ajax e do Single Page.
- **Ajax**: Forma para atualizar um conteúdo de uma página sem atualizar a página toda.
- **Single Page**: Aplicações de página única, utilizando o conceito do Ajax

Resumo

- **JavaScript**:
 - não é Java;
 - não foi criado pelo W3C;
 - é uma linguagem de programação client-side;
 - é utilizada para controlar o HTML e o CSS para manipular comportamentos na página;
 - não é mantido pelo W3C, ele é uma linguagem mantida pela ECMA;

chega de bla bla bla Vamos começar a programar

abra o seu editor de preferência

Executando um programa JavaScript

- O motivo pelo qual o JavaScript é tão popular é ser relativamente fácil de ser acrescentado a uma página WEB.
- Tudo o que você precisa fazer é incluir pelo menos um elemento HTML script na página para especificar "text/javascript" para o atributo `type` e adicionar qualquer JavaScript que quiser.
- JavaScript funciona imediatamente, incluindo um bloco de scripting e já está em ação.

A tag script

- Geralmente o elemento `script` é adicionado ao elemento `head` de uma página por que é mais fácil de realizar a manutenção.
- Alguns autores recomendam a colocação dos elementos `script` na parte de baixo de um documento, para permitir que o resto da página web seja carregada primeiro antes do `script`.
- Independente da abordagem que você usar: coloque seus scripts sempre no elemento `head` ou na parte de baixo do elemento `body`.

Executando um programa JavaScript

- Você pode colocar a quantidade de código que quiser dentro desta tag. O navegador executará assim que tiver feito o download;

```
<!DOCTYPE html>
<html>
<head>
    <title> Exemplo de JavaScript </title>
    <script type="text/javascript">
        </script>
</head>
<body>
</body>
</html>
```

Comentários

- A linguagem JavaScript aceita dois tipos de comentários:
 - Comentário em **linha** ou em **bloco**;

```
<script type="text/javascript">
    //comentário em uma linha

    /*
        comentário
        em
        linhas
    */
</script>
```

Atenção: todos os comentários JavaScript são visíveis no código fonte da página!

Executando um programa JavaScript

- Nem todos os scripts existentes dentro de páginas web são JavaScript.
- A tag de abertura do elemento `script` contém um atributo definindo o tipo do script.
- É uma forma de identificar o conteúdo codificado.
- Entre outros valores permitidos para o atributo `type` estão:
 - `text/ecmascript`;
 - `text/jscript`;
 - `text/vbscript`
 - `text/vbs`

Executando um programa JavaScript

- `text/ecmascript`:** Especifica que o script será interpretado como ECMAScript baseado no padrão de scripting ECMA-262;
- `text/jscript`:** Uma variação de ECMAScript que a Microsoft interpreta no Internet Explorer.
- `text/vbscript` e `text/vbs`:** São interpretados como VBScript da microsoft, uma linguagem scripting completamente diferente.

Executando um programa JavaScript

- Todos esses valores de `type` descrevem o tipo MIME do conteúdo.
- MIME (Multipurpose Internet Mail Extension) é uma forma de identificar como o conteúdo está codificado (`text`) e seu formato específico (`javascript`).
- Versões anteriores da tag `script` recebiam um atributo `language` que era usado para especificar a versão da linguagem (`javascript1.2`). Entretanto este atributo ficou obsoleto em HTML 4.01.

JavaScript x ECMAScript X JScript

- Embora o nome JavaScript tenha se tornado onipresente para scripting baseados em navegadores do lado cliente, apenas **Mozilla** seu popular navegador, o **Firefox**, implementam JavaScript.
- A ECMAScript é, na verdade, uma especificação de scripting do lado cliente que abrange todo o mercado.
- Contudo a maioria dos navegadores reverencia o tipo `text/javascript`.

Criando arquivos externos

- O uso do JavaScript está se tornando mais orientado a objetos e completo.
- Para simplificar desenvolvedores estão criando objetos JavaScript reutilizáveis que podem ser incorporados em muitas aplicações.
- A única forma de compartilhar esses objetos é criá-los em arquivos separados e fornecer um link para cada arquivo na página WEB.

Criando arquivos externos

- Se o código precisa ser alterado posteriormente, é realizado somente em um lugar.
- Atualmente, mesmo o JavaScript mais simples é criado em arquivos scripts separados.
- Para carregar um arquivo js externo na página vamos utilizar o atributo `src` da tag `script`.

Criando arquivos externos

- Para fazer referência a um arquivo JavaScript externo, você precisa usar o atributo **src** na tag `<script>`;

```
<!DOCTYPE html>
<html>
<head>
    <title> Exemplo de JavaScript </title>

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

</head>
<body>

</body>
</html>
```

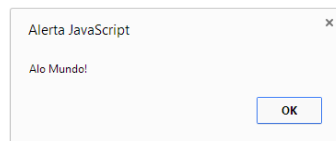
Caixas de diálogo

Caixa de diálogo

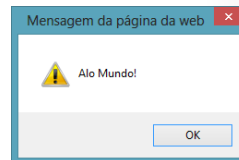
- Caixa de diálogo é uma janela do tipo pop-up que se destina a fornecer informações ou coletar dados do usuário.
- A linguagem JavaScript possui três métodos (ou funções), para o objeto Window, destinadas a criar três tipos de caixa de diálogo.
 - Caixa de alerta
 - Caixa de diálogo de confirmação
 - Caixa de diálogo para entrada de string

Caixa de diálogo

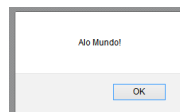
- As caixas de diálogo criadas com JavaScript tem sua apresentação determinada pelo fabricante do navegador:



Chrome



Internet Explorer



Firefox

Caixa de diálogo - Alerta

- `alert()` Box, ou caixa de alerta, destina-se a colocar na interface do usuário uma caixa de diálogo contendo uma mensagem a ele.

```
<script type="text/javascript">  
    alert ("Alo Mundo!");  
    alert ("Alo\nMundo!");  
</script>
```

Caixa de diálogo - Confirmação

- O método `confirm()` do objeto Window destina-se a colocar na interface do usuário uma caixa de diálogo contendo dois botões, normalmente denominados OK e Cancelar, e, ao clicar um deles, ele confirma ou cancela uma determinada ação.
- O retorno do método `confirm()` será:
 - **true**: para escolha ok;
 - **false**: para o cancelar;

```
<script type="text/javascript">
    confirm("Você tem certeza que quer apagar o arquivo? ");
</script>
```

Caixa de diálogo - Confirmação

- O método `prompt()` do objeto Window destinada-se a colocar na interface do usuário uma caixa de diálogo contendo um campo para que ele digite uma string. Essa função admite dois argumentos; `prompt("arg1", ["arg2"]);`
 - **arg1**: é uma instrução ao usuário para o que se espera que ele digite no campo de texto
 - **arg2**: é um valor *default*, facultativo, inserido no campo de texto, normalmente para fornecer uma dica da forma de preenchimento do campo.

```
<script type="text/javascript">
    prompt("Informe a data do seu nascimento:", "dd/mm/aaaa");
</script>
```

Erros e Depuração

Espaços em branco

- Quebras de linhas e espaços em branco, quando inseridos entre nomes de variáveis, nomes de funções, números e entidades similares da linguagem, **são ignorados** na sintaxe JavaScript.
- Contudo, para strings e expressões regulares, tais espaçamentos são considerados.

Sintaxe Válida	Sintaxe Inválida
<code>a=27; e a = 27;</code>	<code>a = 2 7;</code>
<code>document.write("<p> Eu sou \ uma string</p>")</code>	<code>document.write \ ("<p> Eu sou uma string</p>")</code>
<code>document.write ("<p> Eu sou uma string</p>")</code>	<code>document.write("<p> Eu sou uma string</p>")</code>
<code>alert("Olá") e alert ("Olá")</code>	
<code>function(){...} e function () {...}</code>	

Erros e Depuração

- Como alguns dos problemas são quase invisíveis à olho nú, podemos utilizar uma ferramenta que está disponível no Mozilla Firefox, Google Chrome e no Internet Explorer;
- O console de erros não faz parte da linguagem e sim do ambiente.
- **Console de erros:**
 - **Mozilla Firefox:** Ctrl+Shift+J
 - **Google Chrome:** Ctrl+Shift+J ou F12
 - **Internet Explorer:** F12

Atenção: O Console exibe erros de JavaScript, CSS e também erros de cromagem gerados internamente pelo navegador. (extensão defeituosa)

Console de Erros

O console exibe mais que apenas erros:



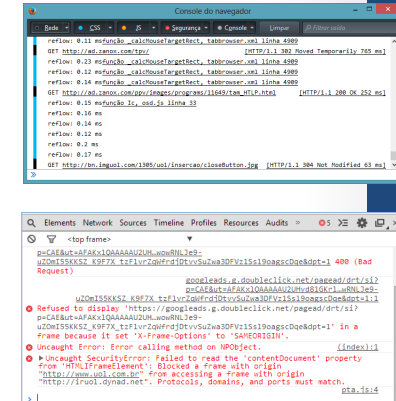
Errors: problemas no seu código que impediram o navegador de continuar o script;



Warnings: Problemas no seu código que o navegador foi capaz de contornar;



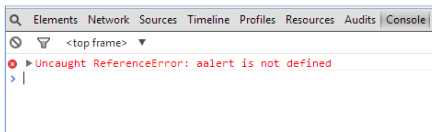
Messages: Anotações de seu código que indicam que esta fazendo normalmente;



Erros e Depuração

- Exemplo de Erro:

```
<script type="text/javascript">
  aalert("Alô Mundo!");
</script>
```



Erros e Depuração - Console

- Você pode usar comandos para enviar mensagens e executar outras tarefas na janela Console do JavaScript;

Comando	Descrição	Exemplo
assert (expressão, message)	Envia uma mensagem se expressão for avaliada como false.	console.assert ((x == 1), "assert message: x != 1");
clear ()	Limpa as mensagens da janela do console, incluindo mensagens de erro de script, e limpa também o script exibido na janela do console. Não limpa o script inserido no prompt de entrada do console.	console.clear ();
count (title)	Envia o número de vezes que o comando count foi chamado para a janela do console. Cada chamada do comando count é identificada exclusivamente pelo title opcional.	console.count ();

Erros e Depuração - Console

Comando	Descrição	Exemplo
debug (message)	Os objetos transmitidos usando o comando são convertidos em um valor de cadeia de caracteres.	<code>console.log("logging message");</code>
dir (object)	Envia o objeto especificado para a janela do console e o exibe em um visualizador de objetos. Você pode usar o visualizador para inspecionar propriedades na janela do console.	<code>console.dir(obj);</code>
dirxml (object)	Envia o nó XML object especificado para a janela do console e o exibe como uma árvore de nós XML.	<code>console.dirxml(xmlNode);</code>
error (message)	Envia message para a janela do console. O texto da mensagem está em vermelho e antecedido por um símbolo de erro.	<code>console.error("error message");</code>

Erros e Depuração - Console

Comando	Descrição	Exemplo
group (title)	Os comandos group* podem facilitar a exibição da saída da janela do console em alguns cenários, como quando um modelo de componente está em uso.	<code>console.log("This is the outer level");</code> <code>console.group("Level 2 Header");</code> <code>console.log("Level 2");</code> <code>console.group();</code> <code>console.log("Level 3");</code> <code>console.warn("More of level 3");</code> <code>console.groupEnd();</code> <code>console.log("Back to level 2");</code> <code>console.groupEnd();</code> <code>console.debug("Back to the outer level 1");</code>
info (message)	Envia message para a janela do console. A mensagem é prefaciada por um símbolo de informação.	<code>console.info("info message");</code>

Erros e Depuração - Console

Comando	Descrição	Exemplo
log (message)	Os objetos transmitidos usando o comando são convertidos em um valor de cadeia de caracteres.	<code>console.log("logging message");</code>
profile (reportName)	chamando esta função inicia um perfil JavaScript CPU	<code>function processPixels() {</code> <code> console.profile("Processing pixels");</code> <code> pixels</code> <code> console.profileEnd();</code> <code>}</code>
profileEnd ()	Interrompe a sessão atual de perfis JavaScript CPU,	
time (name)	Inicia um temporizador que é identificado pelo parâmetro name opcional.	<code>console.time("app start");</code> <code>app.start();</code> <code>console.timeEnd("app start");</code>
timeEnd (name)	Interrompe um temporizador que é identificado pelo parâmetro name opcional. Consulte o comando	

Erros e Depuração - Console

Comando	Descrição	Exemplo
trace ()	Envia um rastreamento de pilha à janela do console. O rastreamento inclui a pilha de chamadas completa e informações como o nome do arquivo, o número da linha e o número da coluna.	<code>console.trace();</code>
warn (message)	Envia message para a janela do console, prefaciada por um símbolo de aviso.	<code>console.warn("warning message");</code>

Referências

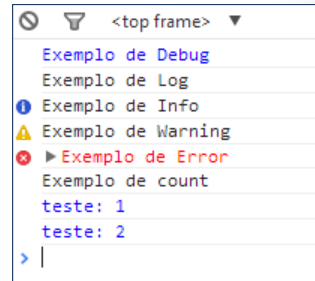
- <https://developers.google.com/chrome-developer-tools/docs/console-api?hl=pt-br>
- <http://msdn.microsoft.com/pt-br/pt-b/library/windows/apps/hh696634.aspx>

Erros e Depuração - Console

- Exemplos

```
<script type="text/javascript">
  console.debug("Exemplo de Debug");
  console.log("Exemplo de Log");
  console.info("Exemplo de Info");
  console.warn("Exemplo de Warning");
  console.error("Exemplo de Error");

  console.log("Exemplo de count");
  console.count("teste");
  console.count("teste");
</script>
```



Variáveis

Variáveis

- Variável é um nome qualquer ao qual se atribui um valor ou dado;
- Uma variável pode conter uma string, número, array, booleano, função, objeto, etc;

Declaração

- Existem três tipos de declarações em JavaScript.
- var:** Declara uma variável, opcionalmente, inicializando-a com um valor.
- let:** Declara uma variável local de escopo do bloco, opcionalmente, inicializando-a com um valor.
- const:** Declara uma constante apenas de leitura.

Classificação

- Uma variável declarada usando a declaração **var** ou **let** sem especificar o valor inicial tem o valor undefined.
- Uma tentativa de acessar uma variável não declarada resultará no lançamento de uma exceção ReferenceError.

```
<script type="text/javascript">
  var a;
  console.log("O valor de a é " + a); //"O valor de a é undefined"
  console.log("O valor de b é " + b); //exception de erro de referência
</script>
```

Variáveis

- O nome da variável é de livre escolha do programador, ressalvadas as seguintes restrições sintáticas:
 - Podem começar com **letra**, **\$** ou **_**
 - Não pode começar com **números**;
 - É possível usar **letras** ou **números**;
 - É possível usar **acentos** e **símbolos**;
 - Não podem conter **espaços**;
 - Não podem ser **palavras reservadas**;
- Escolha nomes que transmitam uma dica do conteúdo da variável.

Escopo de variável

- Quando você declara uma variável fora de qualquer função, ela é chamada de variável **global**, porque está disponível para qualquer outro código no documento atual.
- Quando você declara uma variável dentro de uma função, é chamada de variável **local**, pois ela está disponível somente dentro dessa função.

Escopo de variável

- JavaScript antes do ECMAScript 6 não possuía escopo de declaração de bloco; pelo contrário, uma variável declarada dentro de um bloco de uma função é uma variável local do bloco que está inserido a função.

```
<script type="text/javascript">
  {
    var x = 5;
  }
  console.log(x); // 5
</script>
```

Escopo de variável

- Esse comportamento é alterado, quando usado a declaração `let` introduzida pelo ECMAScript 6.

```
<script type="text/javascript">
{
  let y = 5;
}
console.log(y); // ReferenceError: y não está definido
</script>
```

Palavras Chaves

- Uma restrição para os nomes de variáveis são as palavras-chave JavaScript.

Tabela 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	

Fonte: Aprendendo JavaScript – Shelley Powers

Palavras Chaves

- Devido às extensões propostas à especificação ECMA-262, as palavras da tabela abaixo também são consideradas reservadas.

Tabela 2 – Palavras reservadas da 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	

Fonte: Aprendendo JavaScript – Shelley Powers

Palavras Chaves

- Determinadas palavras específicas de JavaScript implementadas na maioria dos navegadores são consideradas reservadas.

Tabela 3 – Palavras reservadas típicas de navegadores (mais comum)

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

Fonte: Aprendendo JavaScript – Shelley Powers

Diretrizes de nomenclatura

- Funções e variáveis frequentemente começam com letras minúsculas e incorporam uma representação verbal do que a função faz. Ex: `function validaNomeNoRegistro(nome)`
- Muitas vezes, variáveis e funções tem uma ou mais palavras concatenadas em um único identificador. Podemos utilizar um formato chamado **CamelCase**.
- As bibliotecas JavaScript mais novas invariavelmente usam a notação *CamelCase*.
- Também podemos utilizar traços (-) e sublinhados (_) para separar, embora o *CamelCase* torna muito mais legível.

Diretrizes de nomenclatura

- Diversas práticas de nomenclatura, muitas herdadas do Java e outras linguagens de programação podem tornar o código mais fácil de seguir e manter;
- Utilize palavras significativas em vez de algo que tenha juntado rapidamente. Ex: `taxaDeJuros` e não `txJu` ou `tj`;
- Usar plural para coleções; Ex: `var nomesClientes`
- Reservar letras maiúsculas para objetos os torna mais fáceis de diferenciar.

Tipos de Dados

Literais

- Na terminologia JavaScript, a palavra literal designa qualquer dado;
- Existem seis tipos de dados literais conforme descritos a seguir:
 - inteiros;
 - decimais;
 - booleanos;
 - strings;
 - arrays;
 - objetos.

Atenção: a linguagem JavaScript é vagamente tipada, ela não se preocupa com o que as variáveis contém!

Literais

- Seis tipos de dados são os chamados primitivos:
 - Números
 - Strings
 - Booleanos
 - Nulo (null)
 - Indefinido (undefined)
 - Symbol: Um tipo de dado cuja as instâncias são únicas e imutáveis. (ES6)
- Todos os demais são objetos**
 - Matrizes (arrays)
 - Funções
 - Expressões Regulares
 - Objetos

Declaração

- Mesmo não sendo necessário declarar o tipo de dados antecipadamente, ainda é fundamental saber o tipo de dados que a variável irá armazenar;
- Ao contrário da maioria das linguagens de programação, as variáveis da JavaScript podem conter qualquer tipo de dado;
- Para declarar uma variável que pertence ao:**
 - escopo local:** usa-se a palavra-chave **var**.
 - escopo global:** sem uso da palavra-chave **var**;
- É importante declarar as variáveis na primeira linha de código da região para a qual ela será válida;

Declaração e Atribuição

- Exemplo de declaração de variável e atribuição de dados;

```
<script type="text/javascript">

idade=0;
media=0;
nome="";

idade= 28;
media= 40.5;
nome = "Luiz Henrique";
nome = "Luiz Henrique" + "de Angeli"; //concatenando

//utilizando template string
sobrenome = "de Angeli"
nome = "Luiz Henrique ";
console.log("O nome dele é ${nome} ${sobrenome}!");
</script>
```

Convertendo tipos de dados

- Exemplo de conversões de tipos de dados

```
<script type="text/javascript">

inteiro = parseInt("10");
decimal = parseFloat("10.45");
texto = decimal.toString();

inteiro = Number("10");
decimal = Number("10.45");
texto = String(decimal);

</script>
```

Alguns recursos de String e Number

```
<script type="text/javascript">

nome = "Luiz Henrique de Angeli";
console.log(nome.length);
console.log(nome.toUpperCase());
console.log(nome.toLowerCase());

</script>
```

```
<script type="text/javascript">

s = 1575.7;
console.log(s.toFixed(2));
console.log(s.toFixed(2).replace(".",","));
console.log(s.toLocaleString('pt-br', {style: 'currency', currency: 'BRL'}));
console.log(s.toLocaleString('pt-br', {style: 'currency', currency: 'USD'}));
console.log(s.toLocaleString('pt-br', {style: 'currency', currency: 'EUR'}));

</script>
```

Constantes

- Às vezes você desejará definir um valor e tratar apenas como leitura;
- A **constante** pode ter qualquer valor e, por não poder receber um valor posteriormente, é inicializada com seu valor constante quando é definida.
- Da mesma forma que variáveis, uma constante JavaScript tem escopo global e local.

```
<script type="text/javascript">

const MES_ATUAL = 3.5;

</script>
```

Constantes

- “Pontas soltas”

```
<script type="text/javascript">

const PI = 3.14;
PI = 4;

</script>
```

```
<script type="text/javascript">

const objeto = {
  nome : "Luiz",
  idade : 25
};
Object.freeze(objeto);
objeto.idade = 35;

</script>
```

Operadores

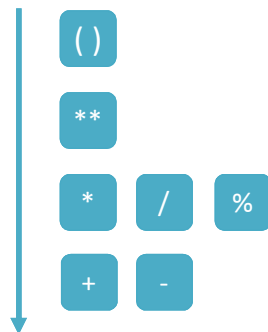
Operadores

- São símbolos, em geral de um ou poucos caracteres, que permitem operações aritméticas, lógicas, etc. Em outras palavras, são usados basicamente para modificar valores de variáveis.
 - Aritméticos;
 - Atribuição;
 - Relacionais;
 - Lógicos;
 - Ternários;

Operadores Aritméticos

Operador	Descrição	Simulação
+	Soma	$3 + 2 \rightarrow 5$
-	Subtração	$3 - 2 \rightarrow 1$
*	Multiplificação	$3 * 2 \rightarrow 6$
/	Divisão	$3 / 2 \rightarrow 1,5$
%	Resto da Divisão	$3 \% 2 \rightarrow 1$
**	Potência	$3 ** 2 \rightarrow 9$
()	Precedência de Operação	$(3 + 2) / 5 \rightarrow 1$

Ordem de Precedência



Operadores de Atribuição

Atribuição	Simplificada
<code>var n = 5</code>	
<code>n = n + 1</code>	<code>n++</code>
<code>n = n - 1</code>	<code>n--</code>
<code>n = n + 5</code>	<code>n+=5</code>
<code>n = n - 5</code>	<code>n-=5</code>
<code>n = n * 5</code>	<code>n*=5</code>
<code>n = n ** 5</code>	<code>n**=5</code>
<code>n = n % 5</code>	<code>n%=5</code>

Operadores Relacionais

Operador	Descrição	Simulação
>	Maior	3 > 2 → true
<	Menor	3 < 2 → false
>=	Maior ou Igual	3 >= 2 → true
<=	Menor ou Igual	3 <= 2 → false
==	Igual	3 == 2 → false
!=	Diferente	3 != 2 → true
===	Identidade ou Igualdade Restrita	2 === 2 → true 2 === '2' → false
!==	Identidade ou Igualdade Restrita	2 !== 2 → false 2 !== '2' → true

Comparação do operador "===" com "=="

Operação	Resultado	Operação	Resultado
' ' == '0'	false	' ' === '0'	false
0 == ''	true	0 === ''	false
0 == '0'	true	0 === '0'	false
false == 'false'	false	false === 'false'	false
false == '0'	true	false === '0'	false
false == undefined	false	false === undefined	false
false == null	false	false === null	false
null == undefined	true	null === undefined	false

Operador "==" não leva em consideração o tipo de dados; e o operador "==="

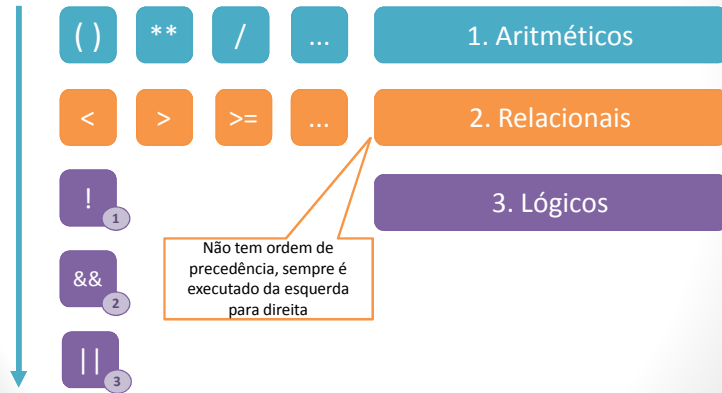
Operadores Lógicos

Operador	Descrição	Simulação
!	Negação	! true → false ! false → true
&&	E ou Conjuncção	true && true → true true && false → false false && true → false false && false → false
	OU ou Disjunção	true && true → true true && false → true false && true → true false && false → false

Operadores Lógicos

Operador	Descrição	Simulação
!	Negação	! 3 == 2 → true
&&	E ou Conjuncção	3 > 2 && 2 > 1 → true 3 > 2 && 2 < 1 → false
	OU ou Disjunção	3 > 2 2 > 1 → true

Ordem de Precedência



Operadores Ternários

Operador	Descrição	Simulação
?:	Ternário	TESTE ? TRUE : FALSE

Operador	Descrição	Simulação
?:	Ternário	media > 6 ? "Aprovado": "Reprovado"
		numero % 2 == 0 ? "Par": "Ímpar"

Exercícios

1. Criar um programa de calculadora onde o usuário digite 2 números de entrada e apresente em uma caixa de alerta as seguintes operações: Soma, Subtração, Multiplicação, Divisão, Média dos Números.
2. Criar um programa que solicite ao usuário digitar um número, apresente a tabuada deste número.
3. Criar um programa para efetuar o cálculo e a apresentação do valor de uma prestação em atraso, utilizando a fórmula:

$$\text{PRESTAÇÃO} = \text{VALOR} + (\text{VALOR} * (\text{TAXA}/100) * \text{TEMPO})$$
4. Faça um programa que leia a área de uma sala a ser pintada, o custo de uma lata de tinta e a área coberta por uma lata de tinta e calcule o custo da pintura da sala.

$$\text{CUSTO} = (\text{ÁREA TOTAL} / \text{ÁREA LATA}) * \text{CUSTO LATA}$$

Estruturas Condicionais

Estruturas Condicionais

- A maneira de tomarmos decisões é usando estruturas especiais chamadas condições e loops;
- As quais ajudam a controlar quais partes de seu programa será executada e quantas vezes esta parte será executada;
- Uma declaração condicional** é um conjunto de comandos que são executados caso uma condição especificada seja verdadeira. O JavaScript suporta duas declarações condicionais: *if...else* e *switch*.

Estruturas Condicionais

- A instrução mais comum é o **IF**, que verifica uma condição e permite a execução de algum código se a condição for satisfeita.

```
if (condição) {
    código condicional
}
```

```
<script type="text/javascript">
    var numero = 10;
    if (numero == 10) {
        alert("Número é igual a 10!");
    }
</script>
```

Estruturas Condicionais

- A instrução IF não oferece nenhuma alternativa para caso a condição não é satisfeita. Esse é o objetivo do **ELSE**

```
if (condição) {
    código condicional
} else {
    código alternativo
}
```

```
<script type="text/javascript">
    var numero = 11;
    if (numero == 10) {
        alert("Número é igual a 10!");
    } else {
        alert("Número é diferente de 10!");
    }
</script>
```

Estruturas Condicionais

O comando **switch** é próprio para se testar uma variável em relação a diversos valores pré-estabelecidos

```
switch (valor) {
    case constante_1:
        código condicional
        break;
    case constante_2:
        código condicional
        break;
    case constante_3:
        código condicional
        break;
    default:
        código condicional padrão
}
```

```
<script type="text/javascript">
    numero = 2;
    switch (numero)
    {
        case 1:
            alert("Um");
            break;
        case 2:
            alert("Dois");
            break;
        default:
            alert("Não encontrado");
    }
</script>
```

Exercício 1



1. Criar um programa de calculadora onde o usuário digite 2 números de entrada e escolha a operação:
 1. Soma
 2. Subtração,
 3. Multiplicação,
 4. Divisão,
 5. Média dos Números

Apresente o resultado da operação escolhida pelo usuário

Exercício 2



- Faça um programa que, dada a idade de um nadador pelo usuário (prompt), classifique-o em uma das seguintes categorias:
 - **Não classificado**: menor que 5 anos
 - **Infantil A**: 5 - 7 anos
 - **Infantil B**: 8 - 10 anos
 - **Juvenil A**: 11 - 13 anos
 - **Juvenil B**: 14 - 17 anos
 - **Sênior**: maiores de 18 anos

Exercício 3



- Faça um programa que leia os valores de 4 notas escolares de um aluno.
 - Calcular a média e apresentar a mensagem “aprovado” se a média for maior ou igual a 7, caso contrário deve solicitar a nota de exame do aluno e calcular a nova média entre a primeira média e a nota do exame.
 - Se a nova média for maior ou igual a 5, apresente “aprovado em exame”, caso contrário apresente a mensagem “reprovado”. Informar junto com a mensagem a média obtida.

Estruturas de repetições

Estruturas de Repetições: Loops

- Existe algumas instruções de loop diferentes, mas elas fazem basicamente o mesmo:
 - Repetir um conjunto de ações até uma condição especifica seja true.
- Loops:
 - While
 - Do-While
 - For
 - For-in

Estruturas de Repetições: **while**

- O loop `while` é o mais simples de todos;
- Tudo o que ele precisa é de uma condição e algum código condicional;

```
while(condição){
    código condicional
}
```

```
<script type="text/javascript">
var contador=10;
while(contador < 10){
    alert(contador);
    contador++;
}
</script>
```

Estruturas de Repetições: **do-while**

- O loop `do-while` se comporta de modo quase idêntico a um loop `while`, com uma diferença importante: o código condicional é colocado antes da condição. Portanto o código condicional é executado pelo menos uma vez;

```
do{
    código condicional
}while(condição);
```

```
<script type="text/javascript">
var contador=1;
do{
    alert(contador);
    contador++;
}while(contador < 10);
</script>
```

Estruturas de Repetições: **for**

- O loop `for` é recomendado para utilizar com números fixos de repetições especificado;

```
for(início;condição;incremento)
{
    código
}
```

```
<script type="text/javascript">
for(var i=0;i<10;i++){
    alert(i);
}
</script>
```

Estruturas de Repetições: **for-in**

- Loops `for-in` são usados para iterar objetos que não sejam arrays;
- O laço `for-in` interage sobre propriedades enumeradas de um objeto, na ordem original de inserção. O laço pode ser executado para cada propriedade distinta do objeto.
- Quando iterar é importante usar o método `hasOwnProperty()` para excluir propriedades que tenham sido herdadas.

```
for(var index in objetos){
  código
}
```

```
<script type="text/javascript">
  pessoa = {nome:"Luiz", idade:28};
  for(var p in pessoa){
    if(pessoa.hasOwnProperty(p))
      console.log(p + " - " + pessoa[p]);
  }
</script>
```

Estruturas de Repetições: **for-in**

- Tecnicamente você também pode utilizar o loop `for-in` em arrays (por que no JavaScript o array também é um objeto) **mas não é recomendado**;
- Índices de arrays somente se tornam propriedades enumeradas com inteiros (integer).
- Não há garantia de que utilizando o laço `for...in` os índices de um array serão retornados em uma ordem particular ou irá retornar todas as propriedades enumeráveis.
- É recomendável utilizar o laço `for` com índices numéricos ou `Array.prototype.forEach()` ou ainda **`for...of`** quando iteragir sobre arrays onde a ordem é importante.

Estruturas de Repetições: **for-of**

- O loop `for...of` percorre objetos iterativos (incluindo Array, Map, Set, o objeto), chamando uma função personalizada com instruções a serem executadas para o valor de cada objeto distinto.

```
for(var valor of lista){
  código
}
```

```
<script type="text/javascript">
  var numeros = [10, 20, 30];
  for (let valor of numeros) {
    console.log(valor);
  }
</script>
```

Exercício



1. Criar um programa de calculadora onde o usuário digite 2 números de entrada e escolha a operação:
 1. Soma
 2. Subtração,
 3. Multiplicação,
 4. Divisão,
 5. Média dos Números

Apresente o resultado da operação escolhida pelo usuário, Faça a leitura das opções até que o usuário escolha a opção sair;

Exercício de Estrutura Condicional e Repetição

- Gerar um número aleatório de 0 a 100 utilizando a função `random()`. Solicite ao usuário para digitar um número até que ele acerte o número ou ultrapasse 10 tentativas;
- A cada número digitado pelo usuário informe com `alert` como esta seu resultado conforme a diferença entre o número gerado e o número digitado pelo usuário:
 - **Gelado:** diferença de 50 ou superior
 - **Frio:** diferença de 40 a 49
 - **Morno:** diferença de 30 a 39
 - **Quente:** diferença de 20 a 29
 - **Fervendo:** diferença de 10 a 19
 - **Queimando:** diferença de 1 a 9

```
<script type="text/javascript">
numero = parseInt(Math.random() * 100);
alert(numero);
</script>
```

Arrays []

`isObject = true`

Arrays []

- Os literais arrays, na sintaxe JavaScript, são os conjuntos de zero ou mais valores, separados por vírgula e envolvidos por colchetes (`[]`).
- Os valores contidos em um array recebem um índice sequencial começando com zero.

```
var variável = [
  "value1",
  "value2"
];

ou

variável[0] = "value1";
variável[1] = "value2";
```

```
<script type="text/javascript">

var frutas = [
  "laranja",
  "pera",
  "goiaba",
  "morango"
];

alert(frutas[0]);

</script>
```

Arrays []

- Os arrays podem ser estruturas muito rápidas;
- **Infelizmente**, o JavaScript não possui nada parecido com esse tipo de array;
- Em vez disso, o JavaScript possui um objeto que tem algumas características semelhantes ao array;
- Ele converte índices de arrays em sequências de caracteres que são usados para criar as propriedades; (Isso é mais lento);
- O JavaScript em si é confuso quanto a diferenciar arrays e objetos. O operador `typeof` (tipo de) diz que o tipo de um array é `object`

Arrays []

Os valores do array que mostramos são strings e devem ser escritos entre aspas. Um array pode conter qualquer tipo de dado da JavaScript, incluindo expressões, objetos e outros arrays.

```
<script type="text/javascript">

  a = 4; b = 12;
  ArrayMisto = [ "laranja", 34, "casa", true, [1,3,5], a+b ];

  alert(ArrayMisto[0]);
  alert(ArrayMisto[1]);
  alert(ArrayMisto[3]);
  alert(ArrayMisto[4][1]);
  alert(ArrayMisto[5]);

</script>
```

Arrays [] – propriedades e métodos

- Variáveis JavaScript podem ser objetos. Arrays são tipos especiais de objetos.
- Devido a isso, você pode ter variáveis de tipos diferentes na mesma matriz.
- O objeto Array tem propriedades e métodos pré-definidos:

Arrays [] – propriedades

Propriedade	Descrição
constructor	Retorna a função que criou o protótipo do objeto Array
length	Define ou retorna o número de elementos em uma matriz
prototype	Permite adicionar propriedades e métodos para um objeto Array

Arrays [] – métodos

Método	Descrição
concat ()	Junta-se duas ou mais matrizes, e retorna uma cópia das matrizes unidas
indexOf ()	Procure a matriz para um elemento e retorna a sua posição
join ()	Junta-se todos os elementos de uma matriz em uma string
lastIndexOf ()	Procure a matriz de um elemento, a partir do final e retorna a sua posição
pop ()	Remove o último elemento de uma matriz e retorna esse elemento
push ()	Acrescenta novos elementos ao final de uma matriz e retorna o novo comprimento
reverse ()	Inverte a ordem dos elementos em uma matriz

Arrays [] – métodos

Método	Descrição
shift ()	Remove o primeiro elemento de uma matriz e retorna esse elemento
slice ()	Seleciona uma parte de uma matriz e retorna a nova matriz
sort ()	Classifica os elementos de uma matriz
splice ()	Adiciona / remove elementos de uma matriz
toString ()	Converte uma matriz em uma string, e retorna o resultado
unshift ()	Adiciona novos elementos para o início de uma matriz, e retorna o novo comprimento
valueOf ()	Retorna o valor primitivo de um array

Veja mais métodos em:

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array#M%C3%A9todos

Arrays [] – propriedades e métodos

```
<script type="text/javascript">
```

```
frutas = ["laranja", "pera", "goiaba", "morango"];
console.log("Tamanho: " + frutas.length);
```

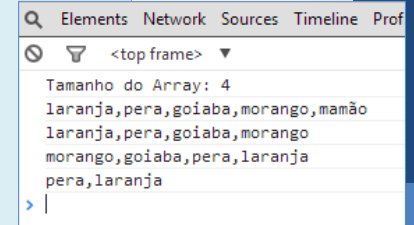
```
frutas.push("mamão");
console.log(frutas.toString());
```

```
frutas.pop();
console.log(frutas.toString());
```

```
frutas.reverse();
console.log(frutas.toString());
```

```
frutas.splice(0, 2);
console.log(frutas.toString());
```

```
</script>
```



Exercício 1

- Criar um programa que solicite ao usuário digitar 10 números. Utilize a função `prompt`. Adicione os números em array utilizando as regras abaixo.
 - Caso o número digitado seja par, remova o número que esta na ultima posição do array e adicione o número digitado no final array.
 - Caso seja impar adicionar o número digitado ao final do array multiplicado por 5 e inverta a ordem dos elementos da matriz.
 - Apresente ao final da leitura de 10 números:
 - A soma dos números contidos no Array;
 - A média dos números contidos do Array
 - O tamanho do array

Números: 1,2,3,4,5,6,7,8,9,10
 1 - Soma: 73
 2 - Média: 14,6
 3 - Tamanho: 5

Exercício 2

- Crie um game de jo-ken-po.
- A cada rodada, o jogador vê o menu: Escolha sua jogada: 1 - Papel 2 - Pedra 3 - Tesoura
- O jogo lê a opção do jogador e verifica se é válida. Se for inválida, o jogador perde a rodada e o jogo acaba. Se for válida, o computador escolhe uma resposta aleatória, que é mostrada ao jogador. Se o jogador ganhar, ele pode jogar mais uma rodada e sua pontuação aumenta. O jogo acaba quando o jogador perde uma rodada. A pontuação total é mostrada no fim do jogo.

Funções ()

isObject = true

Funções : “escrevendo código para depois”

- As funções são como pequenos pacotes de códigos JavaScript prontos para entrar em ação.
- Basta dar um nome ao bloco de código e você poderá chamá-lo a partir de outras áreas de seu programa quando desejar.
- As funções em JavaScript são **objetos**;

```
function função() {
}
```

```
<script type="text/javascript">
function meuNome() {
  alert("Luiz Henrique");
}
</script>
```

Funções : Argumentos

- As funções podem ser projetadas para receber quantos argumentos você quiser de qualquer tipo de dados.

```
function função(argumento1, argumento2){
}
```

```
<script type="text/javascript">
function soma(num1, num2){
  alert(num1, num2);
}
</script>
```

Funções : Instruções de retorno

- Uma função pode **retornar** dados para a instrução que a chamou.
- Para fazer uma função retornar um valor, usamos a palavra-chave **return** seguida do valor que desejamos que ela retorne.

```
function função(argumento1, argumento2){
  return argumento1 + argumento2;
}
```

```
<script type="text/javascript">
function subtracao(num1, num2){
  return num1 - num2;
}
</script>
```

Funções : mantendo suas variáveis separadas

- As variáveis declaradas no **escopo global** podem ser acessadas a partir de qualquer outro código JavaScript que esteja na página web atual.
- Podemos declarar uma variável no **escopo local**, essa variável só existe dentro dos limites da função.

```
variavelGlobal = 10;

function função(){
    var variavelLocal = 10;
}
```

Relembrando: variáveis do **escopo global** declaramos sem a palavra-chave **var**

Funções : mantendo suas variáveis separadas

escopo global

```
<script type="text/javascript">

    function soma(){
        num1=2;
        num2=3;
        return num1 - num2;
    }

    num1 = 10;
    resultado = soma();
    alert(num1);

</script>
```

escopo local

```
<script type="text/javascript">

    function soma(){
        var num1=2;
        var num2=3;
        return num1 - num2;
    }

    num1 = 10;
    resultado = soma();
    alert(num1);

</script>
```

Funções : com parâmetros infinitos

- Uma função pode **receber** parâmetros infinitos .

```
<script type="text/javascript">

function somaTudo(){
    var soma = 0;
    for(var i=0;i<arguments.length;i++){
        soma+=arguments[i];
    }
    return soma;
};

console.log(somaTudo(20));
console.log(somaTudo(20,50,90,35,76,65,50,50,50,14));

</script>
```

Exercício de Funções

- Faça um programa que solicite ao usuário digitado o salário bruto e retorne o salário líquido:
 - Faça uma função para calcular o valor do desconto do INSS
 - Faça uma função para calcular o valor do desconto do IRRF
 - Faça uma função que retorne o salário líquido;



Salário Bruto	Alíquota	Salário Base Calculado	Alíquota	Dedução
até 1.317,07	8%	Até 1.787,77	0%	
de 1.317,08 até 2.195,12	9%	De 1.787,78 até 2.679,29	7,5%	134,08
de 2.195,13 até 4.390,24	11%	De 2.679,30 até 3.572,43	15%	335,03
Acima de 4.390,24	R\$ 482,93	De 3.572,44 até 4.463,81	22,5%	602,96
		Acima de 4.463,81	27,5%	826,15
		R\$ 179,71 por dependente legal		

Exercício de Funções

- **INSS:**
 - $\text{INSS} = \text{SalarioBruto} - \text{ValorTabelaINSS}$
- **IRRF**
 - $\text{ValorBaseIRRF} = \text{SalarioBruto} - \text{INSS} - (\text{Dependentes} * 179.71);$
 - $\text{IRRF} = (\text{ValorBaseIRRF} * \text{PercentualDescontoTabela}) - \text{Dedução}$
- **LÍQUIDO**
 - $\text{SalarioLiquido} = \text{SalarioBruto} - \text{INSS} - \text{IRRF}$

Salário Bruto	Alíquota	Salário Base Calculado	Alíquota	Dedução
até 1.317,07	8%	Até 1.787,77	0%	
de 1.317,08 até 2.195,12	9%	De 1.787,78 até 2.679,29	7,5%	134,08
de 2.195,13 até 4.390,24	11%	De 2.679,30 até 3.572,43	15%	335,03
Acima de 4.390,24	R\$ 482,93	De 3.572,44 até 4.463,81	22,5%	602,96
		Acima de 4.463,81	27,5%	826,15
		R\$ 179,71 por dependente legal		

Funções Conceitos

- **Funções são objetos que podem:**
 - Ser usados dinamicamente no tempo de execução;
 - Ser atribuído a variáveis e ter suas referências copiadas a outras variáveis;
 - Ser passadas como argumentos a outras funções e também ser retornados por outras funções
 - Ter métodos e propriedades próprios

Funções anônimas

- É possível criar uma função usando o construtor `Function` e atribuir a uma variável;

```
<script type="text/javascript">
    soma = new Function("x","y","z","return x + y + z");
    console.log(soma(2,3,1));
</script>
```

Expressões de Função

- Funções literais também são conhecidas como expressões de funções, por que a função é criada como parte de uma expressão;
- Elas lembram *funções anônimas*, pelo fato de não ter nome;

```
<script type="text/javascript">
    add = function (num1, num2){
        return num1 + num2;
    }
    console.log(add(2,4));
</script>
```

Função de Callback

- Funções podem ser passadas como argumentos a outras funções, isso é possível por ser objeto;

```
<script type="text/javascript">

function soma(num1,num2,callback){
    var soma = num1+num2;
    callback();
}

function callback(){
    alert("Função de callback");
}

soma(1,2,callback);

</script>
```

```
<script type="text/javascript">

function soma (num1,num2,callback){
    var soma = num1+num2;
    callback();
}

soma(1,2,function(){
    alert("Função de callback");
});

</script>
```

Função de Callback

- Exemplo utilizando função com variável ou direta.

```
soma = function(numero1, numero2, funcao){
    var resultado = numero1 + numero2;
    funcao(resultado);
};

callback1 = function(resultado){
    console.log("O Resultado é: " + resultado);
};

callback2 = function(resultado){
    console.log("The Result is: " + resultado);
}

soma(15,20, callback1);
soma(10,20, callback2);

soma(50,60,function(res){
    console.log("O resultado da soma é: " + res);
});
```

Função de Callback

- Exemplo de Função de Callback com temporizador:
 - setInterval**: Executa a função infinitamente;
 - setTimeout**: Executa a função somente uma vez;

```
<script type="text/javascript">

function contador(){
    console.count("contador");
}

setInterval(contador, 1000);

</script>
```

```
<script type="text/javascript">

interval = setInterval(function(){
    console.count("contador");
}, 1000);

</script>

clearInterval(contador);
```

Exercício de Funções



- Criar um programa que funcione como um cronometro regressivo, o usuário digitar o tempo que deseja em segundos e o programa diminui o tempo até chegar a zero, mostrar a cada segundo o tempo que falta para zero.
- Criar um programa que tenha um lista de pessoas (array) com o nome e idade. Escreva uma função que receba como parâmetro esta lista e uma função de callback.
 - Na função, percorra a lista de pessoa verificando se a idade é menor que 18 anos.
 - Quando encontrar pessoa com menor de 18 anos, execute a função de callback, passando a lista e o índice desta pessoa;
 - Na função de callback remova o índice recebido do array e informe com uma mensagem o nome da pessoa excluída;
 - Apresente a lista de pessoas atualizada;

Retornando Funções

- Funções também podem ser usadas como valor de retorno;

```
<script type="text/javascript">

    instalador = function(){
        var passo = 0;
        return function(){
            passo++;
            return passo;
        }
    }

    proximo = new instalador();
    console.log(proximo());
    console.log(proximo());

</script>
```

Funções autodefiníveis

- Funções podem ser definidas dinamicamente e ser atribuídas a variáveis;
- Se você escrever uma função e atribuí-la a uma variável que já armazenava outra função, você sobrescreverá a função antiga;

```
<script type="text/javascript">

    meuNome = function(){
        alert("Luiz Henrique");
        meuNome = function(){
            alert("Luiz Henrique de Angeli");
        }
    }

    meuNome();
    meuNome();

</script>
```

Funções imediatas

- O padrão de função imediata é uma sintaxe que lhe permite executar uma função tão logo ela seja definida;
- Ajuda a encapsular certa quantidade de trabalho que você queira realizar, eliminando qualquer traços de variáveis globais;

```
<script type="text/javascript">

    (function(){
        var dias = ["Dom", "Seg", "Ter", "Qua", "Qui", "Sex", "Sab"];
        var hoje = new Date();
        var mensagem = "Hoje é " + dias[hoje.getDay()];
        alert(mensagem);

    })();

</script>
```

Funções imediatas - Parâmetros

- Passando parâmetros para a função imediata;

```
<script type="text/javascript">

    (function(nome){
        var dias = ["Dom", "Seg", "Ter", "Qua", "Qui", "Sex", "Sab"];
        var hoje = new Date();
        var mensagem = "Olá " + nome + " Hoje é " +
            dias[hoje.getDay()];
        alert(mensagem);

    })("Luiz Henrique");

</script>
```

Funções imediatas - Retorno

- Recebendo retorno de uma função imediata;

```
<script type="text/javascript">

    soma =(function(num1, num2) {
        return num1 + num2;

    } (1,2));

    console.log(soma);

</script>
```

Exercício de Funções



- Criar um programa onde o usuário deve digitar uma idade (prompt). Criar uma função que receba esta idade:
 - Caso a idade seja maior ou igual a 18 anos, retorne uma função que solicite o nome e cpf da pessoa;
 - Caso a idade seja menor que 18 anos, retorne uma função que solicite o nome, nome do responsável e RG.

DOM

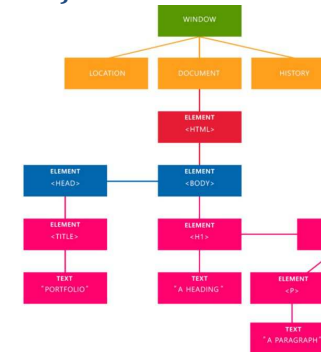
O que é o DOM?

- O DOM (Document Object Model) é uma interface que representa como os documentos HTML e XML são lidos pelo seu browser.
- Após o browser ler seu documento HTML, ele cria um objeto que faz uma representação estruturada do seu documento e define meios de como essa estrutura pode ser acessada.
- É possível acessar e manipular o DOM utilizando JavaScript.

O que é o DOM?

- O objeto *document* que é responsável por conceder ao código Javascript todo o acesso a **árvore DOM** do navegador Web.
- Portanto, qualquer coisa criado pelo navegador Web no modelo da página Web poderá ser acessado através do objeto *document*.
- Usa-se o DOM principalmente para atualizar uma página Web ou quando se quer construir uma interface de usuário avançada.
- Com o DOM pode-se mover itens dentro de uma página ou criar efeitos CSS bastante interessantes sem precisar nem mesmo recarregar a página.

Representação



Fonte: <https://tableless.com.br/entendendo-o-dom-document-object-model/>

Manipulando o DOM

Window

- O objeto *window* representa uma janela que contém um elemento DOM; a propriedade *document* aponta para o documento DOM document carregado naquela janela.
- Acessando link é possível verificar a documentação do *window*.
<https://developer.mozilla.org/pt-BR/docs/Web/API/Window>

DOM

- O DOM possui muitos métodos, são eles que fazem a ligação entre os nodes (elementos) e os eventos;
- Através do objeto *document* pode-se ter acesso a um grande número de propriedades.
- Acessando link é possível verificar a documentação do *document*.
<https://developer.mozilla.org/en-US/docs/Web/API/Document>

Exemplo 1

window.document.write()

```
<script type="text/javascript">
window.document.write("Olá Mundo");
window.document.write(window.document.charset);
window.document.write(window.navigator.appname);
window.document.write(window.document.URL);

</script>
```

Acessando os elementos no DOM

Métodos de Acessos

- Por Marca
- Por ID
- Por Nome
- Por Classe
- Por Seletor

Acessando os elementos no DOM

Por Marca

```
<p> Primeiro Paragrafo </p>
<p> Segundo Paragrafo </p>

<script type="text/javascript">

var p1 = window.document.getElementsByTagName('p')[0];
window.document.write(p1.innerText);

var p2 = window.document.getElementsByTagName('p')[1];
p2.style.color = 'blue';
p2.style.backgroud = 'red';
p2.innerHTML = "novo <b>texto</b>";
window.document.write(p2.innerHTML);

</script>
```

Acessando os elementos no DOM

Por ID

```
<p id='p1'> Primeiro Paragrafo </p>
<p id='p2'> Segundo Paragrafo </p>
<div id='divMsg'></div>

<script type="text/javascript">

var divMsg = window.document.getElementById('divMsg');
divMsg.style.background = 'green';
divMsg.innerHTML = 'mensagem na div';

</script>
```

Acessando os elementos no DOM

Por Nome

```
<p name='p1'> Primeiro Paragrafo </p>
<p name='p2'> Segundo Paragrafo </p>
<div name='divMsg'></div>

<script type="text/javascript">

var divMsg = window.document.getElementsByName('divMsg')[0];
divMsg.style.background = 'green';
divMsg.innerHTML = 'mensagem na div';

</script>
```

Acessando os elementos no DOM

Por Classe

```
<p class='paragrafo'> Primeiro Paragrafo </p>
<p class='paragrafo'> Segundo Paragrafo </p>
<div name='box'></div>

<script type="text/javascript">

var divMsg = window.document.getElementsByClassName('paragrafo')[0];
divMsg.style.background = 'green';
divMsg.innerHTML = 'mensagem na div';

</script>
```

Acessando os elementos no DOM

Por Seletor (novo)

```
<p class='paragrafo'> Primeiro Paragrafo </p>
<p class='paragrafo'> Segundo Paragrafo </p>
<div id='box'></div>

<script type="text/javascript">

var p1 = window.document.querySelector('#p1');
p1.style.background='red';

var p2 = window.document.querySelectorAll('p')[1];
p2.style.background='blue';

</script>
```

Eventos

- Eventos Dom (Dom Events) são utilizados para notificar o código de novidades durante a navegação do usuário.
- Cada evento é representado por um objeto que é baseado na interface *Event*.
- Eventos podem representar desde interações básicas do usuário (cliques, rolagem da página...) até notificações automáticas de novidades no DOM.

Lista dos Eventos: <https://developer.mozilla.org/pt-BR/docs/Web/Events>

Eventos: on....

```
<style> .... </style>
<div id="divArea" onclick="clicar()" onmouseenter="entrar()"
onmouseout="sair()"> Div de Teste </div>
<script type="text/javascript">
var a=window.document.getElementById('divArea');
function clicar(){
    a.innerText = "clicou";
}

function entrar(){
    a.innerText = "entrou";
    a.style.background = "red";
}

function sair(){
    a.innerText = "saiu";
    a.style.background = "blue";
}
</script>
```

Eventos: listener

```
<style> .... </style>
<div id="divArea"> Div de Teste </div>
<script type="text/javascript">

var a=window.document.getElementById('divArea');
a.addEventListener('click', clicar);
a.addEventListener('mouseenter', entrar);
a.addEventListener('mouseout', sair);

function clicar(){}

function entrar(){}

function sair(){}

</script>
```

Acessando elementos de formulário

```
<input type="number" id="numero1">
<input type="number" id="numero2">
<button type="button" onclick="somar()">S</button>
<div id="resultado"></div>
<script type="text/javascript">

function somar(){
    var numero1 = document.getElementById("numero1").value;
    var numero2 = document.getElementById("numero2").value;
    var resultado = numero1 + numero2;
    document.getElementById("resultado").innerHTML = resultado;
}

</script>
```

Exercício

1. Agora refaça o exercício da calculadora, porém, utilizando os recursos HTML, CSS e JavaScript.

128			
	C	BKS	/
7	8	9	*
4	5	6	-
1	2	3	+
	0	.	=



Exercício

1. Faça o jogo do campo minado, a matriz deve ser 10 x 10 e deve conter 10 bombas;

Tentativas: 10
Recorde: 10
Nome: Luiz Henrique

Minas: 100 Bombas: 20

