Curso de JAVASCRIPT

2019

Instrutor: Márcia Maria Tognetti Correa

E-mail: marciac@unicamp.br

Capitulo 01 – MUNDO JAVASCRIPT

➔ O que é JavaScript

JavaScript é uma linguagem de programação baseada em scripts. Trata-se de uma linguagem interpretada, ou seja, o código-fonte é lido e executado diretamente pelo navegador no momento que é carregado através de uma página web. Por ser uma linguagem interpretada não é gerado um arquivo executável. A linguagem Javascript foi inventada por Brendan Eich, da Netscape com parceira com a Sum Microsystems em 1995 (lançada em 1996) com a finalidade de proporcionar interatividade ao desenvolvimento de páginas Web.

JavaScript consegue interagir com, praticamente, todos os elementos de uma página HTML, trabalhar com variáveis, gerar resultados, alterar a aparência de elementos e o melhor, sem a necessidade de ficar recarregando a página. Existem aplicativos inteiros feitos somente usando JavaScript, e ouso dizer que estes aplicativos passarão a ser cada vez mais comuns com o passar do tempo e a evolução desta linguagem.

A saída de um script em *JavaScript* é HTML e o código-fonte é inserido nas páginas web.

O *JavaScript* é uma linguagem orientada a eventos. Ações que ocorrem na página web, sejam pela interação do usuário ou eventos gerados pelo sistema (navegador, sistema operacional, etc.), podem ser tratados como um evento pelo *JavaScript*. Exemplos de eventos:

- Clique num botão da página web
- Preenchimento de um campo de formulário
- Carregar uma página web no navegador

É importante citar que tanto a Netscape como a Microsoft desenvolveram interpretadores JavaScript para serem hospedados no servidor, fazendo com que seja possível rodar Javascript no lado do servidor.

Em tese, para conseguir desenvolver aplicações Web com Javascript, é necessário apenas um navegador.

➔ Funcionalidades Gerais do JavaScript

- ✓ **interação com formulários**: com o Javascript é possível acessar campos e valores digitados num formulário HTML e fazer a validação dos mesmos realizando cálculos e sugestões de preenchimento dos campos de dados.

- ✓ **interação com linguagens dinâmicas**: podemos usar Javascript com outras linguagens de programação Web.

- ✓ **Conformidade com os padrões Web**: dois conceitos surgiram com o conceito de desenvolvimento com conformidade com os padrões Web: Javascript não obstrutivo e da melhoria progressiva.

  Desenvolver Javascript não obstrutivo significa:

1) o conteúdo da página deve estar funcional (ou seja, estar dentro dos padrões Web de desenvolvimento);

2) usar javascript com o objetivo de melhorar a usabilidade da página (o javascript não é mais usados para colocar efeitos espetaculares na página, sem utilidade);

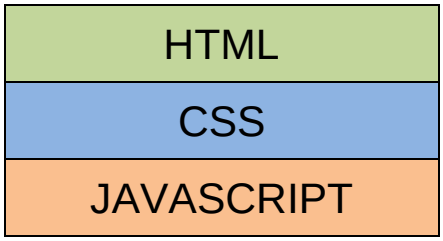
3) não inserir scripts dentro da página HTML. Usar arquivo externo (utilização de separação de camadas de desenvolvimento. O Javascript é mantido na camada de comportamento, não invadindo a camada de estruturação do conteúdo (HTML) e nem a camada de apresentação (CSS)).

O conceito de melhoria progressiva significa (segundo esse conceito, o desenvolvimento de uma página Web pode ser feita em três fases):

Fase 1- estruturação dos conteúdos HTML em páginas que deverão ser acessados por qualquer dispositivo;
Fase 2 – desenvolver a apresentação da página (CSS) a fim tornar agradável a utilização da aplicação para o usuário;
Fase 3 – desenvolver codificação Javascript introduzindo interatividade à página, melhorado a experiência do usuário com a aplicação.

➔ JavaScript e os padrões WEB

É a terceira camada do bolo das tecnologias padrões da web, duas das quais (HTML e CSS). Dentro de uma mesma página Web podemos encontrar HTML, CSS e código JavaScript.



HTML	É a linguagem de marcação usada para estruturar o conteúdo de uma página Web tais como parágrafos, cabeçalhos, tabelas de conteúdo, imagens e vídeos.
CSS	É a linguagem de regras de estilo usada para desenvolver o design de uma página HTML. No CSS conseguimos definir cores de fundo e fontes, tipos e tamanhos de fontes, hierarquia de menus, posicionamentos de elementos de páginas tais como imagens, texto e tabelas.
JAVASCRIPT	É a linguagem de programação que possibilita a criação de conteúdo que atualiza dinamicamente, controla multimídias, imagens animadas, arquivos, menus, etc.

A programação em camadas é fundamental na construção de páginas Web!

Desenvolver aplicações segundo o princípio da separação em camadas de desenvolvimento implica escrever os códigos específicos de cada camada em arquivos separados e "linká-los" enter sí. Citamos quatro vantagens:

1) Elimina a necessidade de repetição de códigos em páginas diferentes;
2) Propicia a reutilização de trechos de códigos em projetos diferentes;
3) Facilita a correção de erros no código;

4) Agiliza a manutenção e compreensão do código feita pelo desenvolvedor.

As três camadas ilustradas acima funcionam muito bem juntas.
Vejamos o exemplo abaixo:

Código HTML:

```
<html>
<head>
<title>Untitled</title>
</head>
<body>
   <h1>Márcia Correa</h1>
</body>
</html>
```

Código CSS:

```
h1 {
  font-family: 'helvetica neue', helvetica, sans-serif;
  text-align: center;
  border: 2px solid rgba(0,0,200,0.6);
  background: rgba(0,0,200,0.3);
  color: rgba(0,0,200,0.6);
  box-shadow: 1px 1px 2px rgba(0,0,200,0.4);
  border-radius: 10px;
  display: inline-block;
  padding: 3px 10px;
}
```

Código JAVASCRIPT:

```
<script>
var para = document.querySelector('h1');

para.addEventListener('click', atualizarNome);

function atualizarNome() {
   var nome = prompt('Digite outro nome, por favor: ');
   para.textContent = 'Novo Nome: ' + nome;
}
</script>
```

Veja o resultado:

Capitulo 02 – MODULARIZAÇÃO E ORGANIZAÇÃO DO CÓDIGO JAVASCRIPT

Em programação quando falamos de modularidade de codificação, estamos falando de definir pequenos códigos com responsabilidades bem definidas que viabilizem a manutenção e reutilização deste código em outras aplicações.

"*Dividir para consquistar*" – famoso conceito usado pelo imperador César nos traz a idéia que a modularização em programação viabiliza um maior controle da aplicação em sua totalidade.

A modularidade permite expandir o código sem comprometer as funcionalidades da aplicação. Nem sempre a ineficiência de um código é por falta de testes. E, sim por dificuldade de manutenção e desenvolvimento da própria lógica.

Os benefícios da modularização são:

- Dividir problemas grandes em vários problemas menores, de baixa complexidade.
    - Número pequeno de variáveis
    - Poucos caminhos de controle (caminhos do início ao fim)
- Possibilidade de utilizar soluções gerais para classes de problemas em lugar de soluções específicas para problemas particulares.
    - Reusabilidade
    - Solucionar uma única vez o problema
- Permite delimitar o escopo (nível de abrangência) de variáveis.
    - Variáveis locais
- Evita a repetição, dentro de um mesmo algoritmo, de uma seqüência de ações em diferentes pontos.

Para fazer bem uma modularização de código é necessário:

1) Conhecer a linguagem que será utilizada;
2) Conhecer exatamente o que deseja-se implementar/codificar – objetivo da aplicação;
3) Definir uma organização da aplicação. Pensar em blocos de código (em componentes);
4) Pensar como esses componentes irão interagir entre sí na aplicação;
5) Pensar em modularidade implicar em não referenciar objetos/estados globais e sim passar como argumentos da função construtora (facilita os testes);
6) Cada bloco/módulo poderá estar num arquivo distinto, separado da aplicação (arquivos com poucas linhas);
7) Colocar comentários no seu código!
8) Planejar/organizar a integração dos módulos implementados;
9) Elaborar a gama de testes manuais ou automatizados dos módulos e da aplicação completa.

Para aprender a modularizar, é necessário praticar!

Capitulo 03 – INTRODUÇÃO A LINGUAGEM JAVASCRIPT

A linguagem Javascript usada autalizamente chama-se ECMA-262 versão 5, lançada em 2009. ECMA é a sigla de European Computer Manufacturers Association. Associação fundada em 1961 dedicada a padronização de sistemas computacionais.

Em 1998, a ISO (International Organization for Standardization) reconheceu a normativa do Javascript oficialmente.

É importante ficar bem claro que os scripts Javascript podem rodar tanto do lado cliente como do lado do servidor de uma aplicação Web.

O navegador Web é onde ficar armazenados os scripts do lado do cliente. Ele hospeda os objetos (janelas, menus, *pop-up*, caixa de diálogo, frames, *cookies*, etc).

No lado do servidor, os scripts armazenados são objetos que representam as requisições ao servidor, clientes, arquivos e mecanismos de bloqueio e compartilhamento de dados.

Antes de iniciarmos nossos estudos sobre a sintaxe é importante dizer que a linguagem Javascript suporta programação Orientada a Objetos (OO). Embora o Javascript simule muitos dos fundamentos OO, não é possível usar todos os conceitos OO tais como herança e encapsulamento.

Os objetos da linguagem Javascript podem ser agrupados em três categorias:

1) **Objetos internos**: strings, arrays, datas, etc;
2) **Objetos de ambiente**: objeto window e document;
3) **Objetos personalizados**: desenvolvidos pelo desenvolvedor.

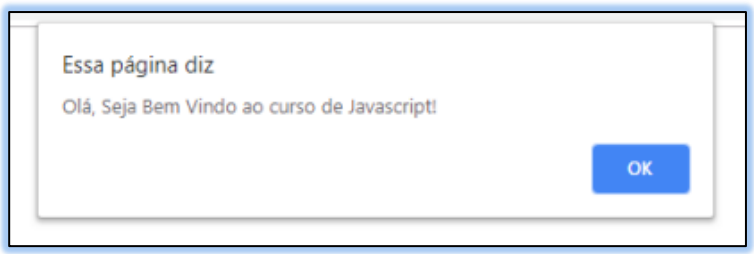
Iniciaremos agora, a introdução aos recursos e sintaxa Javascript. Não podemos esquecer que o Javascript é *case sensitive*, ou seja, os principais comandos devem ser usados em **minúsculo**!

➔ CAIXAS DE DIÁLOGO

Caixa de diálogo é uma janela tipo pop-up que fornece ou coleta informações para e do usuário.

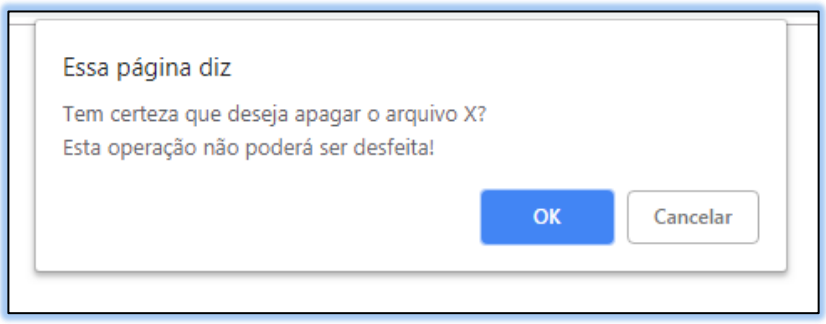
Temos três tipos de caixa: alerta, confirmação e entrada de dados (string). Todos eles são métodos do objeto **window**.

ALERT




```
<script>
window.alert("Olá, Seja Bem Vindo ao curso de Javascript!");
alert("OUTRA FORMA DE USAR O ALERT - Olá, Seja Bem Vindo ao curso de Javascript!");
</script>
```

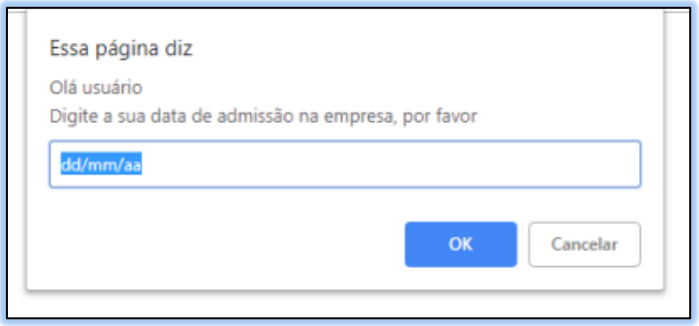
CONFIRM




```
<script>
confirm ("Tem certeza que deseja apagar o arquivo X?\nEsta operação não poderá ser desfeita!");
</script>
```

O botão **OK** é o 1 (true) e o **CANCELAR** é o 0 (false).

PROMPT




```
<script>
prompt ("Olá usuário\nDigite a sua data de admissão na empresa, por favor","dd/mm/aa");
</script>
```

➔ HTML DENTRO DO JAVASCRIPT

É possível colocar código HTML dentro do Javascript utilizando o objeto **document**. Vejamos um exemplo:

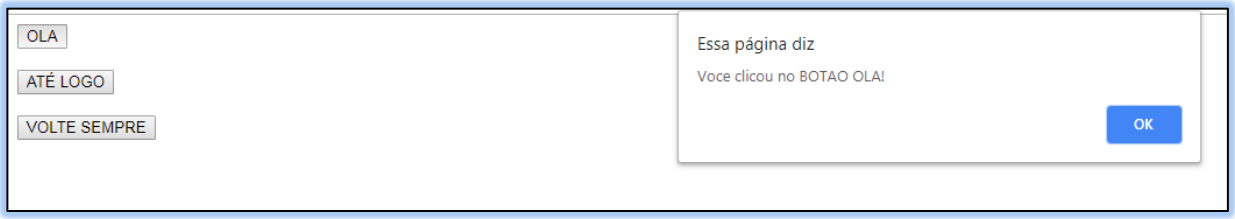

```
<script>
document.write("<font face='tahoma' size='14' color='#0000ff'><b>Curso de Javascript</b></font>");
</script>
```

➔ USANDO EVENTOS DO JAVASCRIPT

Eventos são muito usados em Javascript e ajudam a criar a interatividade em uma aplicação Web. Exemplos de eventos “nativos” do Javascript:

Evento	É disparado…
click	quando é pressionado e liberado o botão primário do mouse, trackpad, etc.
mousemove	sempre que o cursor do mouse se move.
mouseover	quando o cursor do mouse é movido para sobre algum elemento.
mouseout	quando o cursor do mouse se move para fora dos limites de um elemento.
dblclick	quando acontece um clique duplo com o mouse, trackpad, etc.
DOMContentLoaded	quando o DOM do documento está totalmente carregado (mais sobre isso num tutorial futuro).
load	quando todo o documento (DOM e recursos externos como imagens, scripts, etc) está totalmente carregado.
keydown	quando uma tecla no teclado é pressionada.
keyup	quando uma tecla no teclado é liberada (depois de pressionada).
scroll	quando há “rolagem” (scroll) num elemento.

Vejamos um exemplo:




```
<button type="button" onclick="alert('Voce clicou no BOTAO
OLA!')">OLA</button><br><br>
<button type="button" onclick="alert('Voce clicou no BOTAO ATÉ LOGO!')">
ATÉ LOGO</button><br><br>
<button type="button" onclick="alert('Voce clicou no BOTAO VOLTE SEM-
PRE!')">VOLTE SEMPRE</button>
```

O evento usado no exemplo é o ONCLICK (evento acionado ao clicar em um botão).
Agora, para escrever código Javascript numa página HTML, existem três formas de fazer:

- **Inline** – inserindo o código Javascript diretamente no *body* da página HTML (prática não recomendada por ferir a programação em camadas);
- **Incorporado** - inserindo o código Javascript no *head* da página HTML, usando a tag <script>XXX</script>;
- **Externo** – colocando o código Javascript num arquivo com extensão JS (prática altamente recomendada porque está dentro do conceito de programação em camadas).

```
<script type="text/javascript" src="js/meu-arquivo.js"></script>
```

Para o desenvolvedor colocar comentários no código JavaScript, faça:

- Comentário de uma linha: // comentário Y ou <!-- comentário Z;
- Comentário com várias linhas:

```
/* linha comentário 1
   linha comentário 2
   linha comentário 3
*/
```

➔ DECLARAÇÃO DE VARIÁVEIS, ESPAÇO EM BRANCO E LITERAIS EM JAVASCRIPT

Variável é uma posição, frequentemente localizada na memória capaz de reter e representar um dado qualquer. É importante dar nomes significativos as variáveis de trabalho assim como definir o seu tipo: inteiro, real, caracter, string, etc. Ainda sobre o nome de uma variável em Javascript pode usar uma letra ou um número. Evitar usar o cifrão ($) para não haver confusão com códigos específicos das bibliotecas Javascript.

Segue lista das palavras chaves e reservadas em Javascript que devem ser EVITADAS na nomenclatura de variáveis Javascript:

- Palavras-chaves

break	export	super
case	extends	switch
catch	finally	this
class	for	throw
const	function	try
continue	if	typeof
debugger	import	var
default	in	void
delete	instanceof	while
do	new	with
else	return	yield

- Palavras-reservadas

abstract	float	synchronized
boolean	goto	throws
byte	int	transient
char	long	volatile
double	native	
final	short	

Sobre a declaração de variáveis Javascript, é importante definir o seu nome, tipo e escopo: GLOBAL ou LOCAL.

As variáveis em Javascript podem contar qualquer tipo de dado ao longo do programa, ou seja, você pode declarar uma variável chamada valor como sendo inteira a no meio do programa, você

pode mudá-la para string com a simples atribuição de uma string para a variável valor. Veja o exemplo:

```
<script>
    valor = 100;
    alert(valor);
    valor = “Hoje é segunda-feira. Dia ensolarado!”;
    alert(valor);
</script>
```

A variável valor começou como inteiro e acabou como string!

Toda variável de programação tem o seu ESCOPO que definirá se a variável será global ou local. É muito importante entender bem a definição de global e local!

Uma variável é GLOBAL quando o seu conteúdo poderá ser acessado em qualquer parte do programa. E é LOCAL quando seu conteúdo for reconhecido somente dentro da parte onde ela foi declarada.

Em Javascript usamos a palavra reservada VAR para designar a declaração de uma variavel LOCAL.

- Variável **GLOBAL**

numero = 235 (a partir da atribuição de um valor inteiro, fica definido que a variável numero é do tipo inteiro)
nome = “Márcia Tognetti Correa” (a partir da atribuição de um valor string, fica definido que a variável nome é do tipo string)
receita = 1234.78 (a partir da atribuição de um valor real, fica definido que a variável receita é do tipo real)

```
<script>
    valor = 10;
    funcaoPrimeiroTeste  = function() {
      valor = 20;
      alert(“Exibindo resultado da função Primeiro Teste = ”+  valor);
    }

    funcaoSegundoTeste  = function() {
      alert(“Exibindo resultado da função Segundo Teste = ”+  valor);
    }

funcaoPrimeiroTeste();
funcaoSegundoTeste();
</script>
```

Nesse exemplo, a variável valor é GLOBAL!! Em ambas funções, o conteúdo de valor é 20!! Foi inicializado com 10, mas na funcaoPrimeiroTeste, o seu conteúdo foi alterado para 20 e assim permaneceu ate o final do exemplo. Por isso, em ambas as funções, o conteúdo de valor exibido no alert é 20.

- Variável **LOCAL**

Agora, se desejamos que a variável valor, dentro da funcaoPrimeiroTeste tenha escopo local, precisamos colocar a palavra var! Veja o exemplo:

```
<script>
    valor = 10;
    funcaoPrimeiroTeste  = function()  {
      var valor = 20;
      alert(“Exibindo resultado da função Primeiro Teste = ” + valor);
    }

    funcaoSegundoTeste  = function()  {
      alert(“Exibindo resultado da função Segundo Teste = ” + valor);
    }

funcaoPrimeiroTeste();
funcaoSegundoTeste();
</script>
```

No exemplo acima a variável valor, fora da funcaoPrimeiroTeste, tem escopo GLOBAL. Vemos que dentro da funcaoPrimeiroTeste há a declaração de uma variável local valor (sei que é local porque na sua declaração tem a palavra var). Para o Javascript, são duas variáveis distintas. O alert da funcaoPrimeiroTeste exibe o conteúdo 20 (exibe o valor da sua variável local). E o alert da funcaoSegundoTeste exibe o conteúdo 10 (exibe o valor da variável global).

Agora, vamos analisar o exemplo abaixo:

```
<script>
numero = 72;
funcaoExemplo = function () {
      alert(numero);
      var numero = 100;
      alert(numero);
}

funcaoExemplo();
</script>
```

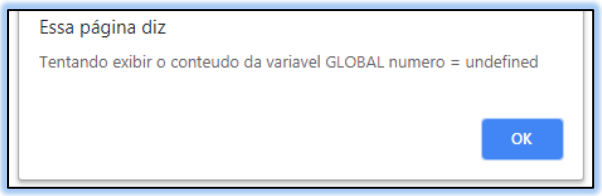
A pergunta que deve ser respondida é: qual o valor que o 1º alert exibirá na página como conteúdo da variável numero? Se sua resposta foi VALOR INDEFINIDO, está correto! Isso ocorre porque para o Javascript o escopo da variável numero é local, pois tem uma variável numero definida dentro da funçãoExemplo. O Javascript até sabe que existe uma variável glogal numero (definida fora do escopo da função), mas não conhecer o seu valor, retornando um valor indefinido.

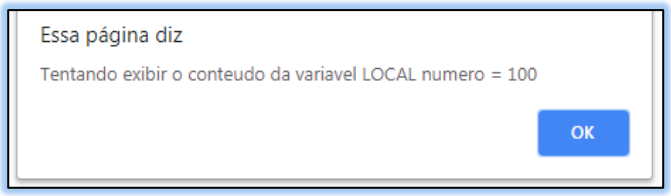
Para evitar isso, sugere-se que as variáveis sejam declaradas logo no inicio da função e usar a palavra resevada var. Veja o exemplo:

```
<script>
numero = 72;
funcaoExemplo = function () {
  alert("Tentando exibir o conteudo da variavel GLOBAL numero = " + numero);
  var numero = 100;
  alert("Tentando exibir o conteudo da variavel LOCAL numero = " + numero);
}
```

```
funcaoExemplo();
</script>
```

Veja e analise os resultados do script acima:





O esperando era que o 1º alert exibisse o conteúdo da variável glogal numero que é 72. E o que ocorreu foi que seu valor é UNDEFINED. Isso ocorreu porque existe dentro da função uma declaração LOCAL de uma variável numero. O Javascript ate “sabe” que existe uma numero GLOBAL, mas não reconhe o seu conteúdo, retornando conteúdo indefinido. Para evitar isso, basta declarar as variáveis locais logo no inicio da função e fazendo outra função, para nesse exemplo, exibir o conteúdo da variável global:

```
<script>
numero = 72;
funcaoExibeGlobal = function () {
  alert("Tentando exibir o conteudo da variavel GLOBAL numero = " + numero);
}

funcaoExemplo = function () {
  var numero = 100;
  alert("Tentando exibir o conteudo da variavel LOCAL numero = " + numero);
}
funcaoExibeGlobal();
funcaoExemplo();
</script>
```

Na sintaxe Javascript temos o **null**, **undefined** e **NaN**. Iremos a seguir definir cada um destes elementos:

Elemento	***Explicaçao***	***Exemplo***
null	Uma variável que não tem conteúdo - indica ausência de valor.	var numero = null;
undefined	Uma variável é indefinida quando é declarada e não inicializada.	var numero;
NaN	Valor não numérico.	var numero = 3.14 / “Educorp”;

➔ TRABALHANDO COM OPERADORES EM JAVASCRIPT

Em Javascritp podemos trabalhar com operadores (os convencionais a toda linguagem de programação) e expressões. Expressão em Javascript é tudo o que gera algum tipo de resultado e pode ser atribuído a uma variável ou função.

Vejamos alguns operadores comuns em Javascript:

Operador	***Explicaçao***
+ - * / %	Soma, subtração, multiplicação, divisão, resto da divisão
+	Concatenação de Strings
< > >= <=	Maior, menor, maior ou igual, menor ou igual
++ --	Incrementar, decrementar
== !=	Igualdade, desigualdade
=== !==	Identidade, não identidade
&& ||	AND, OR
?:	Retorna true ou false em uma expressão condicional
=	Atribuição de conteúdo a uma variável
.	Acessar propriedade ou método de um objeto
()	Chamando uma função
[]	Indexação de vetores (array)
instanceof	Retorna o tipo do objeto
new	Criação de um objeto
typeof	Retorna o tipo de dado de uma variável
void	Retorna um valor indefinido de uma função

Sobre a precedência dos operadores em Javascript é idêntica ao que aprendemos em matemática:

Precedëncia	***Operador***
1º	. [] new
2º	()
3º	**++ --**
4º	*** / %**
5º	**+ -**
6º	**< > >=** **<=** instanceof
7º	== != === !==
8º	&&
9º	||
10º	?:

Operações matemáticas:

Operador	***Explicação***
+	Soma
-	Subtração
*****	Multiplicação
/	Divisão
%	Resto da divisão
++	Increment
--	Decrement

Operadores de comparação:

Operador	***Explicação***	***Exemplo***
==	Igualdade	var x = 27 x == 27 (retorna true) x == 21 (retorna false)
===	Identidade	var x = 27 x === 27 (retorna true) x === 21 (retorna false)
!=	Desigualdade	var x = 30 x != 27 (retorna true) x != 30 (retorna false)
!==	Nao identidade	var x = 30 x !== 27 (retorna true) x !== “30” (retorna false)
>	Maior que	var x = 30 x > 20 (retorna true) x > 100 (retorna false)
<	Menor que	var x = 30 x < 20 (retorna false) x < 100 (retorna true)
>=	Maior que ou igual a	var x = 30 x >= 20 (retorna true) x >= 100 (retorna false)
<=	Menor que ou igual a	var x = 30 x <= 20 (retorna false) x <= 100 (retorna true)

Operadores lógicos:

Operador	***Explicação***	***Exemplo***
&&	AND	var x = 27 var y = 10 (x > y) && (y < 10) (retorna false)
||	OR	var x = 27 var y = 10 (x > y) || (y < 10) (retorna true)
!	NOT	var x = 27 ! (x > 30) (retorna true)

Trabalhar numa expressão matemática com AND ou OR seguirá a mesma lógica usada em outras linguagens de programação. Veja a tabela abaixo, supondo que A = 10 e B = 100):

Variável A	***Operador lógico***	***Variavel B***	***Resultado***
(A > 5)	AND	(B <= 100)	TRUE
(A > 5)	AND	(B < 100)	FALSE
(A < 5)	AND	(B > 10)	FALSE
(A < 5)	AND	(B < 100)	FALSE

(A > 5)	OR	(B <= 100)	TRUE
(A > 5)	OR	(B < 100)	TRUE
(A < 5)	OR	(B > 10)	TRUE
(A < 5)	OR	(B < 100)	FALSE
	NOT	(B <= 100)	FALSE
	NOT	(A < 5)	TRUE

Operadores de atribuição:

Operação	***Explicação***
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /+ y	x = x / y
x <<= y	x = x << y
x &= y	x = x & y

E para concluir o assunto OPERADORES, vamos exemplificar o uso do condicional"**?:**":

```
<script>
  var meuNome;
  var liberacaoAcesso = "Acesso liberado, Sr/Sra " + (meuNome ? meuNome : "usuário desco-
nhecido!");
  alert(liberacaoAcesso);
</script>
```

A sintaxe do operador "?:" é valor = (expressão) ? operando1 : operando2;
Se a expressão for TRUE, o retorno será o operando. Caso contrário, será o operando2.

➔ TRABALHANDO COM ESTRUTURAS DE CONTROLE EM JAVASCRIPT

Em Javascritp temos as seguintes estruturas de controle:

Declaração	***Finalidade***
function	declaração de uma função
return	Retorno o valor de uma função
if... else	Criação de comando condicional
switch	Criação de comando condicional
case	Usando comando condicional switch
break	Usando no switch para validação de valores
default	Usando no switch para valores padrões
for	Criação de comando de repetição
continue	Reiniciar um comando de repetição
while	Criação de comando de repetição
do... while	Criação de comando de repetição
for... in	Criação de comando de repetição em um objeto
throw	Sinalização de erros
try... catch... finally	Tratamento de erros

Vejamos os exemplos:

- ✓ **Exemplo de function**

```
<html>
<head><title></title>
<script>
  function calculaRaizQuadrada(n) {
    return Math.sqrt(n);   }
</script>
</head><body>
<script>
  alert("Raiz Quadrada de 5 = " + calculaRaizQuadrada(5));
</script>
</body>
</html>
```

- ✓ **Exemplo de if... else**

```
<html>
<head><title></title>
<script>
function funcaoBomDia() {
  alert("Bom dia pra voce!");  }

function funcaoBoaTarde() {
  alert("Bom tarde pra voce!");  }

function funcaoBoaNoite() {
  alert("Boa noite. Durma bem!");  }

function funcaoComoVai() {
  alert("Como voce está hoje?");  }

</script></head>
<body>
<script>
  var opcao = 3;
  if (opcao == 1)
    funcaoBomDia();
  else if (opcao == 2)
    funcaoBoaTarde();
  else if (opcao == 3)
    funcaoBoaNoite();
  else funcaoComoVai();
</script></body></html>
```

✓ **Exemplo de switch**

```
<html><head><title></title>
<script>
function funcaoBomDia() {
  alert("Bom dia pra voce!");  }

function funcaoBoaTarde() {
  alert("Bom tarde pra voce!");  }

function funcaoBoaNoite() {
  alert("Boa noite. Durma bem!");  }

function funcaoComoVai() {
  alert("Como voce está hoje?");  }
</script>
</head><body>
<script>
  var opcao = 3;
  switch (opcao) {
    case 1 : funcaoBomDia();   break;
    case 2 : funcaoBoaTarde(); break;
    case 3 : funcaoBoaNoite(); break;
    case 4 : funcaoComoVai();  break;
    default: alert("Opção invalida!"); }
</script></body></html>
```

✓ **Exemplo de while**

```
<html>
<head><title></title>
<script>
function SequenciaNumerica() {
  var resultado = "Sequencia numérica de 1 a 10\n", contador = 1;
  while (contador <= 10) {
   resultado += contador + " ";
   contador++;
  }
  alert(resultado);
}
</script></head>
<body>
<script>  SequenciaNumerica(); </script>
</body></html>
```

- ✓ **Exemplo de “repeat”**

```
<html>
<head><title></title>
<script>
function SequenciaNumerica() {
  var resultado = "Sequencia numérica de 1 a 10\n", contador = 1;
  do  {
    resultado += contador + " ";
    contador++; }
  while (contador <= 10);
  alert(resultado);
}
</script></head>
<body>
<script>  SequenciaNumerica(); </script>
</body></html>
```

- ✓ **Exemplo de for**

```
<html>
<head><title></title>
<script>
function SequenciaNumerica() {
  var resultado = "Sequencia numérica de 1 a 10\n", contador = 1;
  for (contador = 1;contador <= 10;contador++)
    resultado += contador + " ";

  alert(resultado);
}
</script></head>
<body>
<script>  SequenciaNumerica(); </script>
</body></html>
```

- ✓ **Exemplo de for... in**

A declaração for in executa iterações a partir de uma variável específica, percorrendo todas as propriedades de um objeto.

```
<html><head><title>Untitled</title>
<script>
function mostraObj(obj, obj_name) {
  var result = "";
  for (var i in obj) {
    result += obj_name + "." + i + " = " + obj[i] + "\n";
  }
  return result;
}
```

```
</script>
</head><body>
<script>
  var meuCurso = new Object();
  meuCurso.nome = "JAVASCRIPT";
  meuCurso.duracao = "30 horas";
  meuCurso.dataInicial = "26/02/2019";
  alert(mostraObj(meuCurso,"meuCurso"));
</script>
</body></html>
```

➔ TRABALHANDO COM EXCEÇÕES EM JAVASCRIPT

Trabalhamls com exceções numa linguagem de programação quando desejamos tratar possíveis erros no programa ou quando uma funcionalidade não é suportada pelo browser (no caso de aplicações Web).

Em Javascript podemos usar o objeto **Error** para tratar erros. O objeto Error tem duas propriedades message e name:

- Propriedade *message*: retorna uma mensagem com detalhes do erro ocorrido;
- Propriedade *name*: retorna o tipo de erro ocorrido. Seguem os tipos de erros comuns.

E o método toString() que pode ser usado em conjunto com throw e *try... catch... finally*.

- Método toString(): retorna uma string contendo os valores retornados pelas propriedades message e name separados por dois ponto.
- *throw*: o interpretador Javascritp ao encontrar a declaração *throw* indica que houve uma exceção no código e exibe uma mensagem indicando o erro ocorrido. O mais comum é usar o objeto *Error* ou uma mensagem indicando o erro ocorrido.
- *try... catch... finally*: Esse trio forma um bloco de código de tratativa de erro. O bloco *finally* é opcional. Vamos analisar separadamente cada bloco. Vejamos o exemplo:

```
<html><head>
<title>Untitled</title></head>
<body>
<script>
 anoNascimento = prompt("Digite o seu ano de nascimento: ");
 try
 {
   if (anoNascimento <= 0)
   {  throw new Error("O ano não pode ser <= 0!"); }

   if (isNaN(parseInt(anoNascimento)))
     {  throw new Error("Não digitar letras! Digitar somente números inteiros!");  }
 } catch(e)  {
   alert(e.message);
 } finally { alert("Ano correto!"); }
</script></body></html>
```

Caso o ano de nascimento digitado pelo usuário seja <= 0 ou contenha letra(s), será gerado um erro que será exibido através do alert do catch. Caso contrário, o alert do finally será executado. O erro foi tratado e o programa segue seu fluxo normalmente.

Outra construção interessante de exceções, com mais requinte de informações, é:

```
<html><head><title>Untitled</title></head>
<body>
<script>
 var anoNascimento = prompt("Digite o seu ano de nascimento: ","");
  try
 {
   if (anoNascimento <= 0) {
     throw {
       name:"Erro com a entrada de Dados",
       message:"O ano de nascimento precisa ser >= 0";
     }
   }

   if (isNaN(parseInt(anoNascimento)))  {
     throw {
       name:"Erro com a entrada de Dados",
       message:"O ano de nascimento não pode conter letras";
     }
   }
 }  catch (e)  {
   alert(e.name + "\n" + e.message);
 } finally { alert("Finalmente.... ano correto!"); }

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

➔ OBJETOS EM JAVASCRIPT

Praticamente tudo em Javascript é considerado objeto. Objeto é um tipo de dado que armazena dados. Todo objeto tem suas propriedades e seus métodos. Sabemos que todo objeto precisa ser criado no codigo de programação, através do seu construtor.

O construtor Object() é nativo do Javascript e cria um objeto genérico:

var carro = new Object(); (veja que nessa construção o objeto carro ainda não tem propriedades e nem métodos. O objeto carro está vazio).

A seguir, podemos definir suas propriedades:

```
carro.nome = “Fiesta Sedan”;
carro.anoFabricacao = 2015;
carro.cor = “Prata”;
carro.placa = “FXC 2345”;
```

E, a seguir, podemos definir seus métodos:

```
carro.valorKm =  function (distanciaKm, valorCombustivel,qtdeLitrosKm)  {
        var ajudaDeCustoKm  = 0;
        ajudaDeCustoKm  = (distanciaKm;qtdeLitrosKm) * valorCombustivel;
        return ajudaDeCustoKm;  }
```

Novas propriedades ou novos métodos poderão ser usados somente após serem criados e as alterações não influenciarão valores de propriedades anteriores.

Para recuperar o valor de uma propriedade, podemos fazer de duas formas:

```
var nome  = carro.nome  (essa forma é mais usada!)
ou
var nome  = carro[nome].
```

Vamos falar agora sobre o construtor. O que é um construtor de um objeto?

O Construtor é uma função que cria o objeto. É importante salientar que no Javascript não existem classes uma vez que o conceito de Orientação a Objetos no Javascript não é exatamente completo que nem em Java e outras linguagens de programação realmente OO.

Vejamos um exemplo completo de um objeto com seu construtor:

```
<html><head><title>Untitled</title></head>
<body>
<script>
var cubo = {
  medidaLados : 5,
        areaTotal   : 0,
        areaBase    : 0,
        areaLateral : 0
}

function Cubo(m) {
  this.medidaLados = m;
        this.areaTotal = function calculaAreaTotal() {
          x = 6 * Math.pow(this.medidaLados, 2);
                return x
        };

        this.areaBase = function calculaAreaBase() {
          y = Math.pow(this.medidaLados, 2);
                return y
        };

        this.areaLateral = function calculaAreaLateral() {
          z = 4 * Math.pow(this.medidaLados, 2);
                return z
        };
};

var meuCubo = new Cubo(5);
alert("Area Total do Cubo = " + meuCubo.areaTotal() + "\n Area da Base do Cubo = " +
meuCubo.areaBase() + "\n Area da Base Lateral do Cubo = " + meuCubo.areaLateral());
</script>
</body></html>
```

Duas observações referente ao exemplo acima:

1) O construtor sempre começa com a 1º letra em maiúsculo.
2) Os métodos foram criados diretamente dentro da função construtora.

Outra construção para o exemplo acima seria:

```
<html><head><title>Untitled</title></head>
<body>
<script>
var cubo = {
  medidaLados  : 0,
  areaTotal        : 0,
  areaBase         : 0,
  areaLateral      : 0
}

function Cubo(m) {
  this.medidaLados = m;
  this.areaTotal    = calculaAreaTotal;
  this.areaBase    = calculaAreaBase;
  this.areaLateral = calculaAreaLateral;
};

function calculaAreaTotal() {
  return 6 * Math.pow(this.medidaLados, 2);  };

function calculaAreaBase() {
  return Math.pow(this.medidaLados, 2);  };

function calculaAreaLateral() {
  return 4 * Math.pow(this.medidaLados, 2);  };

var meuCubo = new Cubo(5);
alert("Area Total do Cubo = " + meuCubo.areaTotal() + "\n Area da Base do Cubo = " +
meuCubo.areaBase() + "\n Area da Base Lateral do Cubo = " + meuCubo.areaLateral());
</script>
</body></html>
```

Ainda temos outra construção para o mesmo exemplo:

```
<html><head><title>Untitled</title></head>
<body><script>
var cubo = {
  medidaLados : 0,
  areaTotal       : 0,
  areaBase        : 0,
  areaLateral     : 0
}

function Cubo(m) {
  this.medidaLados = m;
```

```
  this.areaTotal    = 6 * Math.pow(this.medidaLados, 2);
  this.areaBase     = Math.pow(this.medidaLados, 2);
  this.areaLateral = 4 * Math.pow(this.medidaLados, 2);
};

var meuCubo = new Cubo(5);
alert("Area Total do Cubo = " + meuCubo.areaTotal + "\n Area da Base do Cubo = " + meu-
Cubo.areaBase + "\n Area da Base Lateral do Cubo = " + meuCubo.areaLateral);
</script></body></html>
```

Todos os exemplos reproduzirão o mesmo resultado! São possibilidades diferentes de construção de um objeto e seu construtor.

Ainda dentro do assunto objetos em Javascript, iremos abordar o objeto Array (vetor de dados).

Sabemos que um array é uma estrutura unidimensional usada para armazenamento de dados ordenados e indexados.

Para criar um vetor em Javascript, usamos o objeto Array:

```
var vet_Numeros = new Array(123,130,156,210,434,1298);
```

Ou criamos também da seguinte forma:

```
var vet_Numeros = new Array;
vet_Numeros[0] = 123;
vet_Numeros[1] = 130;
vet_Numeros[2] = 156;
vet_Numeros[3] = 210;
vet_Numeros[4] = 434;
vet_Numeros[5] = 1298;
```

Observe que o índice começa com **ZERO**!

Para ler um dado de um vetor, será necessário usar um **INDICE**:

```
var indice = 2;
var valor = vet_Numeros[indice];
```

Arrays podem conter dados do tipo objeto:

```
var vet_Dados = new Array(“149.432.323-33”,“Anderson”,”casado”,{idade: 51, dataNiver:
“12;08;1968”,”Campinas”);
```

Para ler um dado do tipo objeto, fazemos:

```
vet_Dados[indice].idade;  ou vet_Dados[indice].idade;
vet_Dados[indice]. dataNiver;  ou vet_Dados[indice].idade;
```

O objeto Array tem as seguintes propriedades:

Propriedade	***Explicação***
constructor	Criar o objeto array (vetor).
prototype	Permite adicionar nova propriedade ou método a um objeto.
length	Retorna a quantidade de elementos existente num array.

Existem métodos interessantes do objeto Array. Na lista abaixo estão os mais usados:

MÉTODOS	EXPLICAÇÃO DO MÉTODO
Concat(arg1,arg2,...,argn)	Acrescenta elementos (os passados como parâmetros) a um array existente
forEach(função(elmento, índice, obj)	Percorre todos os elementos de um array
indexOf(arg)	Retorna o índice de um elemento procurado de um array
lastIndexOf(arg)	Retorna o último índice de um elemento procurado num array
join(arg)	Converte os elementos de um array numa string permitindo acrescentar caracter
pop()	Remove o último elemento de um array retornando o valor removido
push(arg)	Acrescenta um novo elemento a um array retornando a quantidade de elementos existentes no array incluindo o elemento inserido
reverse()	Inverte a odem dos elementos de um array no próprio array
sort(função)	Ordena de forma crescente ou descrescente (dependendo do parâmetro opcional) um array
toString()	Converte os elementos de um array em string

Vejamos alguns exemplos:

- ✓ **Exemplo com o concat**

```
<html><head><title>Untitled</title>
<script>
  function comeco()  {
    var vet = [3,"10",99,10,"Marcia","eca"];
    alert("1o posicao do vetor = " + vet[0]);
    alert("5o posicao do vetor = " + vet[4]);
    var novoVetor = vet.concat(2019,"Guilherme");
    alert("Vetor Original: " + vet);
    alert("Novo Vetor com novos elementos: " + novoVetor);
  }
</script>
</head>
<body onload="comeco();">
</body></html>
```

Somente por curiosidade, segue outra forma de escrever o mesmo código:

```
<html><head><title>Untitled</title>
</head>
<body>
<script>
    var vet = [3,"10",99,10,"Marcia","eca"];
      window.onload = function() {
```

```
    alert("1o posicao do vetor = " + vet[0]);
    alert("5o posicao do vetor = " + vet[4]);
    var novoVetor = vet.concat(2019,"Guilherme");
    alert("Vetor Original: " + vet);
    alert("Novo Vetor com novos elementos: " + novoVetor);
  }
</script></body></html>
```

- ✓ **Exemplo com o forEach**

```
<html><head><title>Untitled</title>
</head>
<body>
<script>
   var vet = [3,99,100,234,1089];
   var resultado = "";
   function ExibeDados(elem, ind, obj) {
     resultado += "vet[" + ind + "] = " + elem + "\n";
   }

   vet.forEach(ExibeDados);
   alert(resultado);
</script></body></html>
```

- ✓ **Exemplo com o indexOf**

```
<html><head><title>Untitled</title>
</head>
<body>
<script>
   var vet = [3,99,100,234,1089];
   var ondeEsta = "";
   function ExibeDados() {
     ondeEsta = "O elemento 100 está na posição " + vet.indexOf(100) + " do vetor";
     alert(ondeEsta);
   }
   ExibeDados();
</script></body></html>
```

Só lembrando que o vetor começa na posição ZERO!

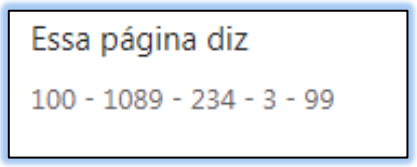
- ✓ **Exemplo com o join**

```
<html><head><title>Untitled</title></head>
<body>
<script>
   var vet = [3,99,100,234,1089];
```

```
    var cadeiaCaracteres = "";
    function ExibeDados() {
      cadeiaCaracteres = vet.join(" - ");
      alert(cadeiaCaracteres);
    }
    ExibeDados();
</script></body></html>
```

- ✓ **Exemplo com o sort**

```
<html><head><title>Untitled</title>
</head>
<body>
<script>
    var vet = [3,99,100,234,1089];
    vet.sort();
    var cadeiaCaracteres = "";
    function ExibeDados() {
      cadeiaCaracteres = vet.join(" - ");
      alert(cadeiaCaracteres);
    }
    ExibeDados();
</script></body></html>
```

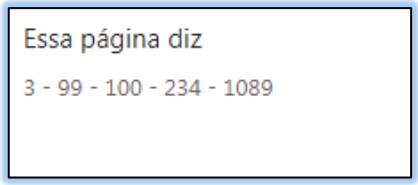


O método sort() faz a ordenação alfabética. Se o vetor for de inteiros, os seus elementos serão convertidos em string.

Para fazer a ordenação numérica de um vetor de números, temos:

```
<html><head><title>Untitled</title>
</head>
<body>
<script>
    var vet = [3,99,100,234,1089];
    vet.sort(function(a,b) {
      return a-b;  })

      cadeiaCaracteres = vet.join(" - ");
      alert(cadeiaCaracteres);
</script></body></html>
```



- ✓ **Exemplo com o toString()**

O método toString() converte cada elemento do vetor numa string e retorna uma lista de dados separados por vírgula.

```
<html><head><title>Untitled</title></head>
<body><script>
    var vet = [3,99,100,234,1089];
    vet.toString()
    alert(vet);</script></body></html>
```

Além do objeto Array, temos vários outros objetos que podem ser interessantes em Javascript. Veremos alguns a seguir:

OBJETOS	EXPLICAÇÃO DO OBJETO
Math	Esse objeto não tem um construtor não podendo ser usado com o operador *new*. Tem as seguintes propriedades: LN10, LN12, LOG10E, PI, SQRT. Os métodos do Math são: abs(x), max(x,y,...,n), min(x,y,...,n), round(x), floor(x), ceil(x), pow(x,y), sqrt(x), random(), exp(x), log(x), sin(x), cos(x)
Number	Esse objeto é um construtor de valores numéricos e pode ser usado com o operador new (var valor = new Number(valor)). Pode-se usar o Number() como função global usada para conversão de tipos de dados. As propriedade do Number() são: MAX_VALUE, MIN_VALUE. Seus métodos são: toString(x), toFixed(x), toPrecision(x) e toExponential(x).

Date	Trata-se de um construtor de datas e horas:
	var d = new Date();
	var d = new Date(milissegundos);
	var d = new Date(data_string);
	var d = new Date(ano, mes, dia, hora, minute, segundo, milissegundo);
	É importante lembrar que a data retornada será a data extraída do SO do computador usado.
	Os métodos do Date() são: getDay(), getFullYear(), getHours(), getMilliseconds(), getMinutes(), getMonth(), getSeconds(), getTime(), getUTCDate() – retorna o dia do mês (1-31), getUTCDay() – retorna o dia da semana (0-7), getUTCFullYear(), getUTCHours(), getUTCMilliseconds(), getUTCMinutes(), getUTCMounth(), getUTCSeconds(), e os métodos set.
	Outros métodos de conversão: toDateString(), toTimeString(), toString().

Document	O objeto document é a página atual que está sendo visualizada nesse momento. As propriedades desse objeto são: alinkColor, area, bgColor, classes, domain, fgColor, Form, ids, Image, link, location, tags, title, URL, vlinkColor.

➔ FUNÇÕES EM JAVASCRIPT

Funções são trechos de códigos que ao serem executados podem retornar um valor para o programa principal. Esses trechos de códigos podem ser usados em diferentes partes do programa principal. É comum criar um arquivo único com as funções que serão usadas na aplicação. Isso facilita a reutilização e manutenção das funções.

Existem três formas de criação de funções em Javascript, vejamos:

1) Função estática:

```
function nomeDaFuncao(pode ter ou não argumentos)  {
    // bloco de comandos
    };
```

É necessário fazer a chamada da função para que ela seja executada. Na declaração da função podemos ou não usar argumentos. Caso seja necessário usar os argumentos, ou seja, enviar parâmetros para a função, estes deverão ser declarados dentro do parêntese da função. Veja o exemplo:

```
<html><head><title>Untitled</title>
<script>
function calculaMediaAluno(n1, n2, n3, n4)  {
  return (n1 + n2 + n3 + n4)/4;  };
</script>
</head>
<body>
<script>
   var media = calculaMediaAluno(2.3, 9.4, 6.6,10);
   alert(media);
</script></body></html>
```

2) <u>Funções anônimas e dinâmicas</u>

```
var objFuncao = new Function(// bloco de comandos)
```

Nesta forma, a função é criada como objeto através do seu construtor Function(). Caso seja necessário, pode-ser passar para o construtor uma lista de argumentos. Nessa forma de declaração não há nome para a função. Na verdade, a função será armazenada numa variável. Veja o exemplo:

```
<html><head><title>Untitled</title>
<script>
  var calculaMediaAluno = new Function("var m = (n1+n2+n3+n4); var media = m/4;
alert('Media do Aluno: ' + media);");
</script></head>
<body>
<button type="button" onclick="calculaMediaAluno(2.3,9.4,6.6,10);">Calcular Media</button>
</body></html>
```

3) <u>Sintaxe literal ou função expressão</u>

```
var objFuncao = function(pode ter ou não argumentos...)   {
// bloco de comandos
};
```

Nesta forma, escolhemos um nome e atribuímos a esse nome a função que será implementada. Veja o exemplo:

```
<html><head><title>Untitled</title>
<script>

  var calculaMediaAluno = function(n1,n2,n3,n4) {
    return alert("Media do aluno: " + (n1 + n2 + n3 + n4)/4);
  };
</script></head>
<body>
<button type="button" onclick="calculaMediaAluno(2.3,9.4,6.6,10);">Calcular Media</button>
</body></html>
```

É importante mostrar que nos exemplos usados, o retorno da função é simples, ou seja, trata-se apenas de um dado único. E se precisarmos retornar mais de uma informação por função? Como proceder?

É possível uma função retornar um array ou um objeto genérico de dados. Veja os exemplos:

➔ Retornando um array de dados

```
<html><head><title>Untitled</title>
<script>

function calculaMediaAluno(n1,n2,n3,n4)
{
  var mediaSimples = (n1 + n2 + n3 + n4)/4;
  var mediaPesos   =  (0.1*n1 + 0.2*n2 + 0.3*n3 + 0.4*n4)/10;
  return [mediaSimples, mediaPesos];
};

</script></head>
<body>
<button type="button" onclick="var resultados = calculaMediaAluno(2.3,9.4,6.6,10);
alert('Media: ' + resultados[0]); alert('Media Pesos: ' + resultados[1])">Calcular Media</button>
</body></html>
```

➔ Retornando um objeto genérico

```
<html><head><title>Untitled</title>
<script>

function calculaMediaAluno(n1,n2,n3,n4)
{
  var mediaSimples = (n1 + n2 + n3 + n4)/4;
  var mediaPesos   = (0.1*n1 + 0.2*n2 + 0.3*n3 + 0.4*n4)/10;
  return {
     mediaSimples: mediaSimples,
     mediaPesos: mediaPesos
  };
};

</script></head>
<body>
<button type="button" onclick="var resultados = calculaMediaAluno(2.3,9.4,6.6,10);
alert('Media: ' + resultados.mediaSimples); alert('Media Pesos: ' + resultos.mediaPesos])">Calcular
Media</button>
</body></html>
```

Vejamos alguns outros exemplos aleatórios:

Exemplo A:

```
<html>
<head>
<title>Untitled</title>
<script>
function soma(a,b)   {
   return a + b;   }
</script>
</head><body>
<script>
var x = 5;
var y = 10;
alert("Resultado da Soma = " + soma(x,y));
</script></body></html>
```

Exemplo B:

```
<html><head><title>Untitled</title>
<script>
var boasVindas = new Function("var meuNome = prompt('Digite seu nome = ',");
          meuNome = (meuNome) ? meuNome : 'usuário desconhecido';
          alert('Bom dia, ' + meuNome);");
</script>
</head>
<body>
<center><br><br>
   <button type="button" onclick="boasVindas();">Chamar Função</button>
</center></body></html>
```

Exemplo C:

```
<html>
<head>
<title>Untitled</title>
<script>
var calculaRaizQuadrada = function(v) {
   return alert("Raiz Quadrada = " + Math.sqrt(v));   };
</script>
</head><body>
<script>
var valor = prompt("Digite um número inteiro qualquer");
calculaRaizQuadrada(valor);
</script></body></html>
```

Retornando ao assunto de escopo de variável, somente lembrando que: o corpo de uma função cria um escopo local para variáveis nela declaradas e, nesse caso, usamos a palavra reservada **var**. Vale lembrar também que os parâmetros recebidos numa função também são escopo local.

As variáveis declaradas fora do escopo de uma função, usando ou não a palavra **var**, pertencem com escopo global e podem ser acessadas por qualquer função.

Vejamos um exemplo:

```
<html><head><title>Untitled</title>
<script>
function calculaRaizQuadrada() {
 // variavel rais é local!
  var raiz = prompt("Digite um número inteiro qualquer");
  alert("Exibindo de dentro da funcao - Raiz Quadrada = " + Math.sqrt(raiz));
}
</script></head><body>
<script>
try {
  calculaRaizQuadrada();
  alert("Exibindo de fora da funcao");
  alert("O dobro da raiz quadrada é: " + 2*raiz);
}
catch(e) {
  alert(e.message);
}
</script></body></html>
```

A variável raiz foi declarada com a palavra reservada **var** significando que o escopo dela é local, ou seja, existe somente dentro da função calculaRaizQuadrada().

Vejamos agora outro exemplo:

```
<html><head><title>Untitled</title>
<script>
function calculaRaizQuadrada()  {
  raiz = prompt("Digite um número inteiro qualquer");
  alert("Exibindo dentro da funcao - Raiz Quadrada = " + Math.sqrt(raiz));
}

</script></head><body>
<script>
try {
  calculaRaizQuadrada();
  alert("Exibindo de fora da funcao - O dobro da raiz quadrada é: " + 2*raiz);
}
catch(e) {
  alert(e.message);
}
</script></body></html>
```

No exemplo acima a variável raiz foi declarada de forma global, sem o uso da palavra **var**. Embora tenha sido declara dentro de uma função, o seu escopo é global por não ter a palavra var.

Ao criar suas variáveis de trabalho é interessante analisar bem o escopo delas dentro da aplicação para poder usar bem o escopo local e global. No entanto, o que temos visto no mercado é a utilização restrita de variavieis globais!

Existem algumas funções globais em Javascript que é interessante comentar:

FUNÇÕES	EXPLICAÇÃO
escape(string)	Retorna uma nova string com caracteres substituídos por sua sequência hexadecimal.
eval(string)	O argumento da função eval() é uma string. Se a string representa uma expressão, eval()avalia a expressão. Se o argumento representa uma ou mais declarações de JavaScript, eval() avalia as declarações.
isFinite(valor)	Retorna true se o valor for um número. Determina se o valor transmitido é um número finito.
isNaN(valor)	Retorna true se valor NÃO for numérico.
Number(valor)	Converte em número o valor passado como argumento da função.
parseFloat(string)	Converte em número real uma string. Caso o 1º caracter não seja um número, a função retornar NaN.
parseInt(string)	Converte em número inteiro uma string. Caso o 1º caracter não seja um número, a função retornar NaN.

➔ TRABALHANDO COM STRINGS EM JAVASCRIPT

Em Javascript, temos o objeto global **STRING** para criar instancias de strings para armazenar textos.

Para criarmos uma string, temos duas formas:

1) var frase = “Curso de Javascript”; (tipo de dado)
2) var frase = new String(“Curso de Javascript”); (criando um objeto do tipo String)

Vejamos as propriedades do objeto String:

PROPRIEDADES	EXPLICAÇÃO
length	Retorna a quantidade de caracteres existentes na string.
prototype	Permite adicionar novas propriedades ou métodos a um objeto já criado.
constructor	Referencia a função que cria o objeto String.

Segue exemplo com a propriedade length:

```
<html><head><title>Untitled</title>
<script>
function tamanhoCampo() {
  var conteudo = document.frmExemplo.txt_Nome.value;
  alert("Exibindo o que foi digitado: " + conteudo);
  var tamanho = conteudo.length;
  alert("Qtde de letras do nome digitado: " + tamanho);
}
</script></head>
<body>
<form name="frmExemplo" action="" method="">
<label>Nome Completo<input type="text" name="txt_Nome" maxlength="60"></label>
<button type="button" onclick="tamanhoCampo();">Validar Nome</button>
</form></body></html>
```

Vejamos alguns dos principais métodos do objeto String:

MÉTODOS	EXPLICAÇÃO
charAt(indice)	Retorna o caracter da string que ocupa a posição definida no parâmetro indice.
concat(string1,string2,...,stringN)	Contatena duas ou mais strings no último parâmetro.
indexOf(stringProcurada)	Retorna o índice da 1º ocorrência da string definida no parâmetro recebido pela função.
match(expressao)	Retorna as ocorrências da string definida no parâmetro recebido que é uma expressão regular. Para usar a função match é necessário conhecer a forma de montar uma expressão regular.
replace(string, novastring)	Encontra uma string igual ao 1º parâmetro e substitui essa string pelo 2º parâmetro recebido. Ambos parametros são obrigatórios.
search(expressao)	Retorna a posição da 1º ocorrência da string contida no parâmetro recebido que é uma expressão regular. Para usar a função search é necessário conhecer a forma de montar uma expressão regular.
slice(inicio, fim)	Cria uma nova string resultante da extração da substring delimitada pelos parâmetros inicio e fim.
substring(indiceA,indiceB)	Essa função é similar ao slice com a diferença que o 1º parâmetro é maior que o 2º. O 1º parâmetro indica a posição a partir da qual a extração deve iniciar. E o 2º parâmetro indica em qual posição deve-se parar a extração.
toLowerCase()	Retorna a string com todos os caracteres em minúsculo.
toUpperCase()	Retorna a string com todos os caracteres em maiúsculo.

Vejamos alguns exemplos clássicos com algumas das propriedades mencionadas no quadro acima:

Exemplo com charAt(índice):

O 1º caracter ocupa o índice zero e o último o length-1.

```
<html><head><title>Untitled</title>
<script>
function PosicaoNome() {
  var nome = document.frmExemplo.txt_Nome.value;
  var indice = Number(document.frmExemplo.txt_indice.value);
  if (indice >= 0 && indice <= nome.length-1) {
    alert("Letra da posição desejada: " + nome.charAt(indice-1));
  } else {
    alert("Digite um indice válido!");
  }
}
</script></head>
<body>
```

```
<form name="frmExemplo" action="" method="">
<label>Nome Completo<input type="text" name="txt_Nome" maxlength="60"></label><br>
<label>Posição Numérica<input type="text" name="txt_indice" maxlength="2"></label><br>
<button type="button" onclick="PosicaoNome();">Verificando Posição na String</button>
</form></body></html>
```

Exemplo com concat:

```
<html><head><title>Untitled</title>
<script>
function JuntaTudo() {
  var tudoJunto = document.frmExemplo.txt_f1.value.concat(" ", docu-
            ment.frmExemplo.txt_f2.value," ",document.frmExemplo.txt_f3.value);
  alert("Frases concatenadas: " + tudoJunto);
}
</script>
</head>
<body>
<form name="frmExemplo" action="" method="">
<label>Frase 1<input type="text" name="txt_f1" maxlength="30"></label><br>
<label>Frase 2<input type="text" name="txt_f2" maxlength="30"></label><br>
<label>Frase 3<input type="text" name="txt_f3" maxlength="30"></label><br>
<button type="button" onclick="JuntaTudo();">Junta Tudo</button>
</form>
</body></html>
```

Exemplo com indexOf:

```
<html><head><title>Untitled</title>
<script>
function ProcuraPedaco() {
  var f = document.frmExemplo.txt_frase.value;
  var p = document.frmExemplo.txt_pedaco.value;
  var i = document.frmExemplo.txt_posicao.value;
  var resultado = f.indexOf(p,i);
  if (resultado >= 0) {
    alert("Posição inicial da substring procurada: " + resultado);
  } else {
    alert("A substring não existe a partir da posição inicial desejada!");
  }
}
</script></head>
<body>
<form name="frmExemplo" action="" method="">
<label>Frase Desejada<input type="text" name="txt_frase" maxlength="50"></label><br>
<label>Posição Desejada<input type="text" name="txt_posicao" maxlength="2"></label><br>
<label>Pedaço da String<input type="text" name="txt_pedaco" maxlength="15"></label><br>
<button type="button" onclick="ProcuraPedaco();">Procura Pedaco</button>
</form></body></html>
```

Exemplo com replace:

```
<html><head><title>Untitled</title>
</head>
<body>
<script>
  var frase = 'Hoje é quinta-feira da 10o semana do ano';
  var novafrase = frase.replace('feira','obaoba');
  alert("Nova frase: " + novafrase);
</script>
</body></html>
```

Exemplo com search(expressão):

```
<html><head><title>Untitled</title>
</head>
<body>
<script>
  var frase = 'Hoje é quinta-feira da 10o semana do ano';
  alert("de H a L: " + frase.search(/[H-L]/));
  alert("1o ocorrencia de qu: " + frase.search('qu'));
</script>
</body></html>
```

Exemplo com substring(indiceA, indiceB):

```
html><head><title>Untitled</title>
<script>
function PegaPedaco() {
  var f  = document.frmExemplo.txt_frase.value;
  var i1 = document.frmExemplo.txt_p1.value;
  var i2 = document.frmExemplo.txt_p2.value;
  var pedaco = f.substring(i1,i2);
  alert("Pedaco da frase da posição : " + i2 + " até a posição " + i2 + " - " + pedaco);
}
</script>
</head>
<body>
<form name="frmExemplo" action="" method="">
<label>Digite uma frase de até 40 letras<input type="text" name="txt_frase" maxlength="40"
            size="40"></label><br>
<label>Inicio da Substring<input type="text" name="txt_p1" maxlength="2"></label><br>
<label>Fim da Substring<input type="text" name="txt_p2" maxlength="2"></label><br>
<button type="button" onclick="PegaPedaco();">Pega Pedaco da String</button>
</form>
</body></html>
```

Exemplo com toUpperCase:

```
<html><head><title>Untitled</title>
</head>
<body>
<script>
  var frase = 'Hoje é QUINTA-FEIRA da 10o semana DO ano';
  alert("Tudo maiúsculo: " + frase.toUpperCase());
</script>
</body></html>
```