



UNIVERSIDADE EDUARDO MONDLANE
FACULDADE DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA

Programação Web e SGC

Introdução ao JavaScript

Docentes: Ruben Manhiça

Maputo, 20 de abril de 2024



Conteúdo da Aula

1. Introdução ao Javascript;
2. Uso e aplicação do Javascript;
3. Sintaxe
4. Exemplos





O que é JavaScript?

- É uma linguagem de programação
 - Desenvolvida pela Netscape®
 - Microsoft® tem o VBScript ⇒ não é tanto utilizado quanto o JavaScript
 - JavaScript também corre no Microsoft Internet Explorer®
 - O código é escrito dentro da página HTML (*client-server*)
 - Simples e Útil (para fazer coisas simples)

Obs.: JavaScript NÃO É Java

- Java entra nas páginas HTML através de *applets*
 - JavaScript é escrito dentro da página HTML





Para que serve “JavaScript”?

- Efeitos visuais
- Efeitos interativos
- Geração dinâmica de conteúdo (*on-the-fly*)
 - Adequações para a resolução do monitor
 - Padronização da apresentação
 - Armazenamento de informações (*cookies*) ⇒ Personalização do conteúdo
- Manipular objetos de interface
 - Janelas, Barra de status, Formulários etc.
- Operações matemáticas e textuais
- Validação de dados de um formulário
- Animações
 - DHTML





Para que serve JavaScript?

- ⇒ Realçar páginas HTML estáticas
- ⇒ Dar Interatividade a formulários
- ⇒ Validar dados no cliente
- ⇒ Interligação entre objetos HTML, Applets Java, ActiveX, e plug-ins no lado do cliente
- ⇒ Aplicações no lado do Servidor





Fatos Sobre JavaScript

- ⇒ Código embutido na página HTML
 - ➔ Não gera programa executável do tipo arquivo exe
- ⇒ Código interpretado pelo navegador
 - ➔ Dependente do navegador
- ⇒ Baseado em objeto
 - ➔ Não aceita herança





Fatos Sobre JavaScript

⇒ Guiada por evento

➔ Respostas a eventos gerados pelos objetos HTML tais como botões e caixas de texto

⇒ JavaScript NÃO É Java!

➔ Sintaxe semelhante, mas conceitos diferentes





Embutindo JavaScript no HTML

- ⇒ O código JavaScript pode também ficar em um arquivo de texto separado da página HTML
- ⇒ O código JavaScript pode ser escrito em qualquer lugar do documento HTML mas é boa prática colocá-lo no cabeçalho do documento (`<Head> ... </Head>`)





Embutindo JavaScript no HTML (v1)

```
<html>  
  <head>  
    <SCRIPT SRC="MyScript.js">  
  </SCRIPT>  
  </head>  
  <body>  
  </body>  
</html>
```





Embutindo JavaScript no HTML (v2)

```
<html>  
  <head>  
    <script type="text/javascript">  
      codigo javascript  
    </script>  
  </head>  
  <body>  
  </body>  
</html>
```





Linguagem JavaScript

⇒ Tipos de dados

- ⇒ number, Boolean, string ,function, object
- ⇒ linguagem fracamente tipada

⇒ Variáveis

- ⇒ global x local
- ⇒ *To var or Not to var?!!*

⇒ Operadores

- ⇒ similares aos das linguagens C++ e Java (ex. $x += y$ é equivalente a $x = x + y$)
- ⇒ Operadores no nível de bits (Bitwise Operators)
- ⇒ Operadores relacionais (comparações)





Tipos de Dados (Exemplos)

```
var length = 16;           // Number
var lastName = "Johnson";  // String
var x = {firstName:"John", lastName:"Doe"}; // Object

var x;                     // Now x is undefined
var x = 5;                 // Now x is a Number
var x = "John";            // Now x is a String
```





Operadores Aritmeticos

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement





Operadores Aritmeticos

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





Operadores aritmeticos(Exemplos)

```
var price1 = 5;  
var price2 = 6;  
var total = price1 + price2;
```

```
Var x = "5" + 2 + 3;  
Resultado: x= 523
```

```
Var y = 5 + 2 +3  
Resultado: y=10
```





Operadores de Comparacao

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Operator	Description
&&	logical and
	logical or
!	logical not





Operadores comparativos (Exemplos)

```
age = Number(age);  
if (isNaN(age)) {  
    voteable = "Error in input";  
} else {  
    voteable = (age < 18) ? "Too young" : "Old enough";  
}
```





Linguagem JavaScript

- ⇒ Controle de fluxo e repetição
 - if ... else, while, do ... while, for, with, switch e case
 - break, continue e rotulada.
 - for ... in
- ⇒ Funções
 - retornam valor, retornam expressão
 - manipulam argumentos extras
 - passagem de parâmetro por valor ou por referência





Fluxo e Repeticoes (Exemplos)

```
if (hour < 18) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}
```

```
switch(expression) {  
    case n:  
        code block  
        break;  
    case n:  
        code block  
        break;  
    default:  
        code block  
}
```





Fluxo e Repeticoes (Exemplos)

```
for (i = 0; i < cars.length; i++) {  
    text += cars[i] + "<br>";  
}
```

```
for (x in person) {  
    text += person[x];  
}
```





Fluxo e Repeticoes (Exemplos)

```
while (i < 10) {  
    text += "The number is " + i;  
    i++;  
}
```

```
do {  
    text += "The number is " + i;  
    i++;  
}  
while (i < 10);
```





Fluxo e Repeticoes (Exemplos)

```
for (i = 0; i < 10; i++) {  
    if (i === 3) { break; }  
    text += "The number is " + i + "<br>";  
}
```

```
for (i = 0; i < 10; i++) {  
    if (i === 3) { continue; }  
    text += "The number is " + i + "<br>";  
}
```





Modelo de Objetos JavaScript

- ⇒ JavaScript organiza hierarquicamente os objetos em uma página
- ⇒ Objetos JavaScript
 - ⇒ Objetos de navegação:
 - window, frame, document, form, button, checkbox, hidden, fileUpload, password, radio, reset, select, submit, text, textarea, link, anchor, applet, image, plugin, area, history, location, navigator..
 - Objetos JavaScript x tags HTML.
 - ⇒ Objetos predefinidos da linguagem
 - nunca são visíveis na página
 - por exemplo: String e Array





Modelo de Objeto JavaScript

⇒ Criando um novo objeto:

- ⇒ `function Livro(titulo, ...) {
 this.titulo = titulo }`

- ⇒ `java = new Livro(java, ...)`

⇒ Objetos de navegação

- ⇒ criando uma janela

 - `window.open(URL, Nome, "height=400,width=600")`

- ⇒ exibindo uma caixa de mensagem

 - `alert(mensagem)`

- ⇒ `location` e `history`





Modelo de Objetos JavaScript

⇒ Objeto Document

➤ `<BODY> ... </BODY>`

➤ propriedades

➤ métodos

⇒ Objeto Form

➤ reflete um formulário HTML em JavaScript

➤ `<form> ... </form>`

➤ Botão

➤ Multiseleção





Modelo de Objetos JavaScript

⇒ Objetos predefinidos

⇒ String

→ ex. length, substring, fontsize, link(URL), toUpperCase

⇒ Array

→ new Array

⇒ Date

→ new Date , getMonth, getCalendarMonth

⇒ Math

→ random, sin, sqrt

⇒ Number

→ new Number(MAX_VALUE)





Modelo de Objetos JavaScript

⇒ Manipulando Eventos

⇒ onClick x onSubmit

→ `<form method="POST" onSubmit="return confirmOrder()">`

⇒ onReset, onChange, onFocus e onBlur, onMouseOver e onMouseOut, onLoad e onUnload, onError, onAbort





Conectividade

- ⇒ Um código JavaScript pode acessar métodos e campos públicos de uma applet Java
- ⇒ A comunicação inicia com uma referência a uma applet:
 - ⇒ **document.applet[0]** (referência à 1ª applet na página)
 - ⇒ **document.applet["AppletName"]** (referência por nome)
- ⇒ Estabelecida a referência, o código JavaScript pode chamar os métodos da applet





JavaScript e Java

- ⇒ Se o objeto myApplet implementa um método público hello [public method **hello()**] esse método pode ser invocado pelo seguinte código:

<form>

<input type="radio" checked="true"

name= "Ola"

onclick="document.myApplet.hello()">

</form>





JavaScript e Java

- ⇒ A passagem de argumentos deve ser feita com atenção pois os tipos de argumentos são diferentes nessas linguagens
- ⇒ Os três tipos básicos string, number e Boolean em JavaScript são diretamente compatíveis com os tipos objeto string, Boolean, e Double em Java
- ⇒ Objetos complexos em JavaScript, como arrays, são refletidos em Java como objetos do tipo **JSObject**





JavaScript e Java

- ⇒ JavaScript também pode acessar métodos e campos estáticos das bibliotecas Java (core packages)

```
var pi = Packages.java.lang.Math.PI;
```

- ⇒ Também é possível construir novos objetos Java dinamicamente, usando o operador **new**

```
var theDate = new java.util.Date()
```





JavaScript e Segurança

- ⇒ “O risco de segurança mais freqüente para máquinas locais acessando a Internet está relacionado à execução de código JavaScript no cliente”
- ⇒ Não há um modo de determinar ou parar a execução de código JavaScript embutido em páginas HTML (se estiver habilitado pelo navegador)
- ⇒ JavaScript não deveria ser usado para criar aplicações responsáveis pelo controle de acesso porque o código é acessível através da janela de fonte e o algoritmo de segurança pode ser facilmente deduzido
- ⇒ Service Attacks, tais como laços infinitos, diálogos modais infinitos e uso total da memória podem bloquear a interface do usuário e requerer que o usuário feche o navegador





JavaScript e Segurança

- ⇒ JavaScript implementa as seguintes características de segurança:
- ⇒ Não é capaz de ler ou escrever arquivos
 - ⇒ Não há acesso a informações do sistema de arquivos
 - ⇒ JavaScript não pode executar programas nem comandos do sistema
 - ⇒ JavaScript não pode fazer conexões de rede com outros computadores, exceto com a máquina de onde a applet foi carregada
 - ⇒ Assinatura eletrônica de objecto



Exemplos

```
<script language="JavaScript">
function ValidaSemPreenchimento(form){
for (i=0;i<form.length;i++){
var obg = form[i].obrigatorio;
if (obg==1){
if (form[i].value == ""){
var nome = form[i].name;
alert("O campo " + nome + " é obrigatório.");
form[i].focus();
return false;
}
}
}
return true;
}
</script>
```

```
<form method="post" action="" onSubmit="return
ValidaSemPreenchimento(this)">
<input type="text" obrigatorio="1" name="texto">
<input type="submit" Value="enviar" />
</form>
```





```
<form>
<SCRIPT LANGUAGE="JavaScript">
var copytoclip=1
function HighlightAll(theField) {
var tempval=eval("document."+theField)
tempval.focus()
tempval.select()
if (document.all&&copytoclip==1){
therange=tempval.createTextRange()
therange.execCommand("Copy")
window.status="Conte'do selecionado e copiado para a -rea
setTimeout("window.status=''",2400);
}
}
function highlight(field) {
    field.focus();
    field.select();
}
function goToURL() { window.location = "javascript:HighlightAll('forms[0].select')"; }
</script>
<textarea name="select" onClick='highlight(this);'>TEXT</textarea><br>
<input type=button value="Selecionar e Copiar" onClick="goToURL()"></p>
</form>
```

Exemplos

FUNÇÕES:

1. AO CLICAR NESTA CAIXA, TODO O TEXTO É SELECIONADO.
2. AO CLICAR NO BOTÃO ABAIXO, ESTE TEXTO SERÁ SELECIONADO E COPIADO PARA A ÁREA DE TRANSFERÊNCIA. UMA MENSAGEM NA BARRA DE STATUS SERÁ MOSTRADA!
* NO MOZILLA FIREFOX, APENAS A FUNÇÃO SELECIONAR FUNCIONA.

Selecionar e Copiar





Alô mundo - versão 1

```
<html>
```

```
<head>
```

```
  <title>Alo!</title>
```

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

```
    alert("Alo mundo!");
```

```
  </script>
```

```
</head>
```


```
<body>
```

```
  ...
```

```
</body>
```

```
</html>
```





Alô mundo – versão 2

```
...  
<head>  
  <title>Alo!</title>  
  <script type="text/javascript" src = "alomundo.js">  
  </script>  
</head>  
...
```

alomundo.html

```
alert("alo mundo");
```

alomundo.js





Sintaxe

- Tudo é case-sensitive, ou seja: **teste** é diferente de **Teste**
- Construções simples: após cada instrução, finaliza-se utilizando um ponto-e-vírgula:

Instrução1;

Instrução2;

- Ex:

```
alert("alo");
```

```
alert("mundo");
```





Sintaxe

- Comentários de uma linha:

```
alert("teste"); // comentário de uma linha
```

- Comentário de várias linhas:

```
/* este é um comentário  
de mais de uma linhas */
```

- Saída de dados: em lugar de usar a função alert, podemos utilizar:

```
document.write("<h1>teste</h1>");
```

- Onde **document** representa a própria página e write escreve no seu corpo.





Variáveis

- Variáveis são usadas para armazenar valores temporários
- Usamos a palavra reservada **var** para defini-las
- Em JS, as variáveis são fracamente tipadas, ou seja, o tipo não é definido explicitamente e sim a partir de uma atribuição (=)

- Ex:

var x = 4; ← Declaração e atribuição de valor

var y; ← Declaração sem atribuição

y = 2; ← Atribuição

alert (x + y);





Alguns tipos

- Números: inteiros e decimais:

var i = 3;

var peso = 65.5;

var inteiroNegativo = -3;

var realNegativo = -498.90;

var expressao = 2 + (4*2 + 20/4) - 3;

- Strings ou cadeia de caracteres:

var nome = "José";

var endereco = "rua" + " das flores";

nome = nome + " maria";

endereco = "rua a, numero " + 3;

← concatenação

← concatenação

← concatenação com
conversão numérica
implícita





Alguns tipos

- Arrays: alternativa para o armazenamento de conjuntos de valores:

```
var numeros = [1,3,5];
```

```
var strNumeros = [];
```

```
strNumeros[0] = "First";
```

```
strNumeros[1] = "Second";
```



```
var vilas= [ ];
```

```
vilas[0] = "Morrumbala";
```

```
vilas[1] = "Morrumbene";
```

```
vilas[2] = "Muidumbe";
```

```
alert("A vila de Cabo-Delegado é " + vilas[2]);
```





Alguns tipos

- Tamanho de um array: usamos a propriedade `length` do próprio array

```
alert(cidades.length);
```

- Último item de um array:

```
alert(cidades[cidades.length-1]);
```





Alguns tipos

- Arrays associativos:
 - baseados também na idéia `array[indice] = valor`
 - O índice/chave de um array associativo é geralmente uma string

```
var idades = [];
```

```
idades["ely"] = 29;
```

```
idades["paulo"] = 20;
```

```
idades["maria"] = 20;
```

```
alert("Minha idade é: " + idades["maria"]);
```





Alguns tipos

- Lógico: tipo que pode ter os valores **true** ou **false**

```
var aprovado = true;
```

```
alert(aprovado);
```





Operador de tipos

- typeof: inspecionar o tipo de uma variável ou valor:

```
var a = "teste";
```

```
alert( typeof a); // string
```

```
alert( typeof 95.8); // number
```

```
alert( typeof 5); // number
```

```
alert( typeof false); // boolean
```

```
alert( typeof true); // boolean
```

```
alert( typeof null); // object
```

```
var b;
```

```
alert( typeof b); // undefined
```





Operador de tipos

- Utilizando typeof podemos ter os seguintes resultados:
 - undefined: se o tipo for indefinido.
 - boolean: se o tipo for lógico
 - number: se for um tipo numérico (inteiro ou ponto flutuante)
 - string: se for uma string
 - object: se for uma referência de tipos (objeto) ou tipo nulo

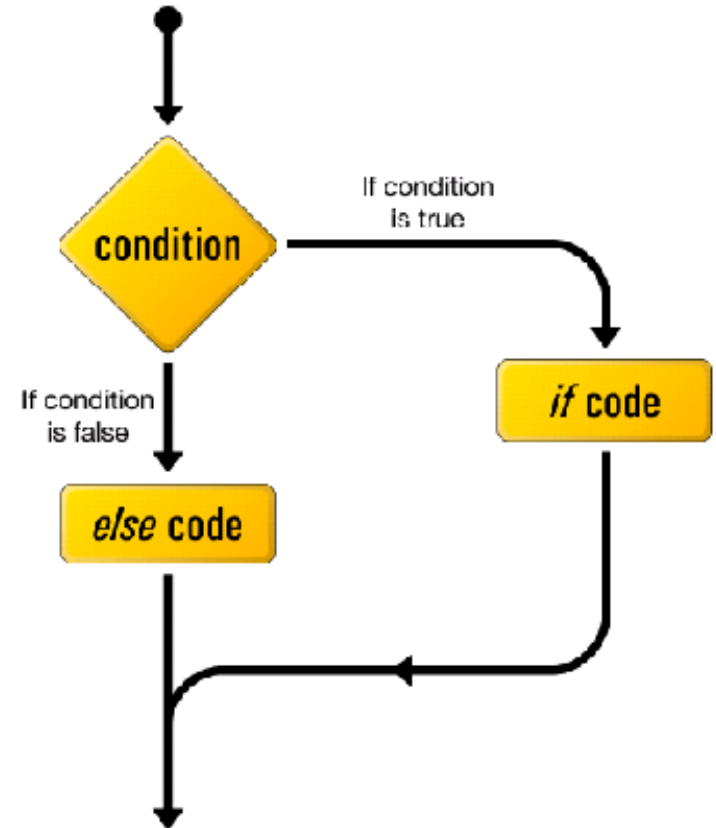


Estruturas de decisão – if e else

- Sintaxe:

```
if (condição) {  
    código da condição verdadeira  
}  
else {  
    código da condição falsa;  
}
```

{ simboliza um início/begin
}
} representa um fim/end





Operadores condicionais e lógicos

>	A > B
>=	A >= B
<	A < B
<=	A <= B
==	A == B
!=	A != B

← A é **igual** a B

← A é **diferente** de B

- && : and
- || : or
- ! : not

```
var idade = 17;  
if (idade >= 16 && idade < 18) {  
    alert("voto facultativo");  
}
```





Estruturas de decisão – if e else

```
if (navigator.cookieEnabled) {  
    alert("Seu navegador suporta cookies");  
} else {  
    alert("Seu navegador não suporta cookies");  
}
```





Estruturas de decisão – Switch

```
switch (expressão) {  
    case valor 1:  
        //código a ser executado se a expressão = valor 1;  
        break;  
    case valor 2:  
        //código a ser executado se a expressão = valor 2;  
        break;  
    ...  
    case valor n:  
        //código a ser executado se a expressão = valor n;  
        break;  
    default:  
        //executado caso a expressão não seja nenhum dos valores;  
}
```





Estruturas de decisão – Switch

```
var idade = 20;
switch (idade) {
    case (29):
        alert("Você está no auge.");
        break;
    case (40) :
        alert("A vida começa aqui.");
        break;
    case (60) :
        alert("Iniciando a melhor idade.");
        break;
    default:
        alert("A vida merece ser vivida, não importa a idade.");
        break;
}
```

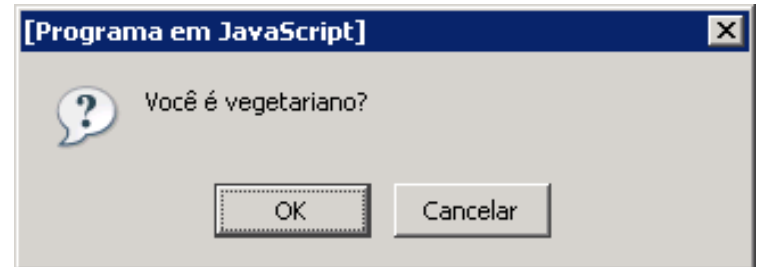


Janelas de diálogo - Confirmação

- Nos permite exibir uma janela pop up com dois botões: **ok** e **cancel**
- Funciona como uma função:
 - Se o usuário clicar em **ok**, ela retorna **true**; em **cancel** retorna **false**
- Ex:

```
var vegetariano = confirm("Você é vegetariano?");
```

```
if (vegetariano == true) {  
    alert("Coma mais proteínas");  
}  
else {  
    alert("Coma menos gordura");  
}
```

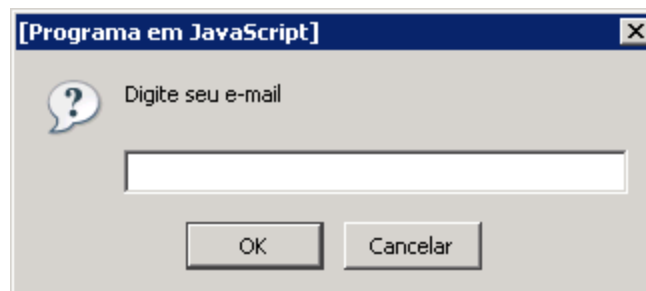




Janelas de diálogo - Prompt

- Nos permite exibir uma janela pop up com dois botões (**ok** e **cancel**) e uma caixa de texto
- Funciona como uma função: se o usuário clicar em **ok** e **prencher a caixa de texto**, ela retorna **o valor do texto**; em **cancel** retorna **null**
- O segundo parâmetro pode ser preenchido como uma sugestão
- Ex:

```
var email = prompt("Digite seu e-mail", "");  
alert("O email " + email + " será usado para spam.");
```





Janelas de diálogo - Prompt

- O que lemos da janela prompt é uma string
- Podemos converter strings para inteiro utilizando as funções pré-definida **parseInt** e **parseFloat**
- **parseInt(valor, base)**: converte uma string para inteiro.
 - O valor será convertido para inteiro e base é o número da base (vamos usar base 10)
- **parseFloat(valor)**: converte uma string para um valor real/ponto flutuante





Janelas de diálogo - Prompt

- Ex:

```
var notaStr = prompt("Qual a sua nota?","");
```

```
var trabStr = prompt("Qual o valor do trabalho?","");
```

```
var nota = parseFloat(notaStr,10);
```

```
var trab = parseFloat(trabStr,10);
```

```
nota = nota + trab;
```

```
alert("Sua nota é: " + nota );
```





Estruturas de repetição - for

- Executa um trecho de código por uma quantidade específica de vezes
- Sintaxe:

```
for (inicio; condicao; incremento/decremento) {  
    //código a ser executado.  
}
```

- Ex:

```
var numeros = [1, 2, 3, 4, 5];  
for (var i = 0; i < numeros.length; i++) {  
    numeros[i] = numeros[i]* 2;  
    document.write(numeros[i] + "<br/>");  
}
```





Expressões compactadas

- Em JS podemos utilizar formas “compactada” instruções:

numero = numero + 1 equivale a **numero++**

numero = numero - 1 equivale a **numero--**

numero = numero + 1 equivale a **numero += 1**

numero = numero - 1 equivale a **numero -= 1**

numero = numero * 2 equivale a **numero *= 2**

numero = numero / 2 equivale a **numero /= 2**





Estruturas de repetição - while

- Executa um trecho de código enquanto uma condição for verdadeira
- Sintaxe:

```
while (condicao) {  
    //código a ser executado  
}
```

Ex:

```
var numero = 1;  
while (numero <= 5) {  
    alert("O número atual é: " + numero);  
    numero = numero + 1;  
}
```





Estruturas de repetição – do...while

- Executa um trecho de código enquanto uma condição for verdadeira
- Mesmo que a condição seja falsa, o código é executado pelo menos uma vez
- Sintaxe:

```
do {  
    //código a ser executado.  
} while (numero <= 5) ;
```

Ex:

```
var numero = 1;  
do {  
    alert("O número atual é: " + numero);  
    numero = numero + 1;  
} while (numero <= 5) ;
```





Funções

- Funções são blocos de código reutilizáveis.
- Elas não são executadas até que sejam **chamadas**
- Podem ter parâmetros de entrada e de saída
- Podemos ter vários parâmetros de entrada separados por vírgulas
- Podemos retornar um valor através da instrução **return**





Funções

- Sintaxe:

```
function nomeDaFuncao() {  
    //códigos referentes à função.  
    ...  
}  
function nomeDaFuncao(p1, p2, p3, ...) {  
    //códigos referentes à função.  
    ...  
}  
function nomeDaFuncao(p1, p2, p3, ...) {  
    ...  
    return p1+p2-p3;  
    ...  
}
```





Funções

- Ex. 1:

```
...  
<a href = "#" onclick = "alo();">Chamar a função</a>  
...
```

← alomundo.html

```
function alo() {  
    alert("Link clicado!");  
}
```

← alomundo.js





Funções

- Ex. 2:

```
...  
<form>  
    <input type = "button" value = "Chamar função" onclick = "alo();" />  
</form>  
...
```

```
function alo() {  
    alert("Link clicado!");  
}
```

← alomundo.html

← alomundo.js





Funções

- Ex. 3: Passando parâmetros

```
...  
<form>  
    <input type = "button" value = "Chamar função"  
        onclick = "saudacao('jose');"/>  
</form>  
...
```

```
function saudacao(nome) {  
    alert("Olá, " + nome);  
}
```

saudacao.html

saudacao.js





Funções

- Ex. 4: Passando parâmetros de campos de formulário

```
...  
<form>  
    <input type="text" name="txtNome" id = "txtNome"/>  
    <input type="button" name="btn_saudacao"  
        onclick = "saudacao(document.getElementById('txtNome').value);"/>  
</form>  
...
```

```
...  
<form name = "frm">  
    <input type="text" name="txtNome"/>  
    <input type="button" name="btn_saudacao"  
        onclick = "saudacao(frm.txtNome.value);"/>  
</form>  
...
```





Funções

- Ex. : retornando valores e escrevendo no documento

```
function soma(v1, v2) {  
    return v1 + v2;  
}
```

```
function soma(v1, v2) {  
    document.write(v1 + v2);  
}
```

← soma.js





Eventos

- São reações a ações do usuário ou da própria página
ou:
- São ocorrências ou acontecimentos dentro de uma página. Ex:
 - Carregar uma página;
 - Clicar em um botão;
 - Modificar o texto de uma caixa de texto;
 - Sair de um campo texto;
 - etc;





Eventos

- **onclick**: ocorre quando o usuário clica sobre algum elemento da página

```
...  
<a href = "#" onclick = "alo();" >Chamar a função</a>  
...
```

- **onload** e **onunload**: ocorrem respectivamente quando o objeto que as possuem são carregados (criados) e descarregados

```
...  
<body onload = "bemvindo();" onunload = "adeus();" >  
...
```





Eventos

```
function bemvindo() {  
    alert("Seja bem vindo.");  
}
```

```
function adeus() {  
    alert("Obrigado pela visita.");  
}
```





Eventos

- **onmouseover**: é acionado quando o mouse se localiza na área de um elemento
- **onmouseout**: ocorre quando o mouse sai da área de um elemento

```
...  
<a href = "#" onmouseover = "mouseSobre();"   
                                onmouseout = "mouseFora();">
```

Passe o mouse

```
</a>
```

```
<div id = "resultado"> </div>
```

```
...
```





Eventos

```
function mouseSobre() {  
    var divResultado = document.getElementById("resultado");  
    divResultado.innerHTML = divResultado.innerHTML + "mouse  
sobre.<br/>";  
}  
  
function mouseFora() {  
    var divResultado = document.getElementById("resultado");  
    divResultado.innerHTML = divResultado.innerHTML + "mouse  
fora.<br/>";  
}
```





Eventos

- **onsubmit:** usado para chamar a validação de um formulário (ao enviar os dados)
- Para validar um formulário, chamamos uma função por nós definida:
 - Ao chamar a função, usamos a palavra reservada return
- A função, por sua vez, deve retornar true ou false, representando se os dados devem ou não serem enviados. Ex:

```
<form name="frmBusca"
      action="http://www.google.com/search"
      method="get" onsubmit = "return validaCampo()">
  Termo: <input type="text" name="q" id = "q" />
  <input type="submit" name="btnBuscar" value="Buscar"/>
</form>
```





Eventos

```
function validaCampo() {  
    var valor = document.getElementById("q").value;  
  
    if ((valor == null) || (valor == "")) {  
        alert("Preencha o campo de busca");  
        return false;  
    }  
    return true;  
}
```





Eventos

- **onfocus**: ocorre quando um controle recebe o foco através do mouse ou do teclado
- **onblur**: ocorre quando um controle perde o foco

...

```
<input type="text" name="txt1" id = "txt1"  
        onfocus = "trataEntrada('txt1')"  
        onblur = "trataSaida('txt1')"/>
```

```
<input type="text" name="txt2" id = "txt2"  
        onfocus = "trataEntrada('txt2')"  
        onblur = "trataSaida('txt2')"/>
```

...





Eventos

```
function trataEntrada(id) {  
    var div = document.getElementById("resultado");  
    div.innerHTML = div.innerHTML + id + " ganhou o  
    foco.<br/>";  
}
```

```
function trataSaida(id) {  
    var div = document.getElementById("resultado");  
    div.innerHTML = div.innerHTML + id + " perdeu o  
    foco.<br/>";  
}
```





Eventos

- **onkeydown** e **onkeypress**: são semelhantes e ocorrem quando uma tecla é pressionada pelo usuário em seu teclado.
- **onkeyup**: é executado quando a tecla é liberada, ou seja, ela foi pressionada e em seguida liberada.

...

```
<input type="text" name="txtOrigem" id = "txtOrigem"  
      onkeydown = "copiaTexto('txtOrigem','txtDestino')"/>
```

```
<input type="text" name="txtDestino" id = "txtDestino" />
```

...





Eventos

```
function copiaTexto(idOrigem,idDestino) {  
    var txtOrigem =document.getElementById(idOrigem);  
    document.getElementById(idDestino).value = txtOrigem.value;  
}
```





Validações de formulários

- Os dados de um formulário devem ser enviados para um servidor.
 - Pode-se suavizar o trabalho de um servidor efetuando-se algumas validações no próprio cliente (navegador) com JavaScript
 - Nota: É importante também haver a validação no servidor.
- A validação com JavaScript serve apenas para amenizar o tráfego de rede com validações simples como campos não preenchidos, caixas não marcadas e etc.





Validações de formulários

- Algumas dicas:
 - Ao se validar um campo, procure sempre obtê-los pelo atributo `id`
 - Quase todos os elementos do formulário possuem sempre um atributo **value**, que pode ser acessado como uma String
 - Para verificar um caractere em especial dentro de um valor, use `[]`, pois as Strings são arrays de caracteres
 - As Strings também possuem um atributo **length** que assim como os arrays, representam o tamanho





Validações de formulários

- Alguns exemplos de validação:
 - Campos de texto não preenchidos
 - Campo de texto com tamanho mínimo e máximo
 - Validação de campo de e-mail
 - Campos com apenas números em seu conteúdo
 - Seleção obrigatória de radio buttons, checkboxes e caixas de seleção





Validações de formulários

- Validação de campo de texto com preenchimento obrigatório:
 - Deve-se validar se:
 - O valor é nulo
 - O valor é uma String vazia
 - O valor é formado apenas por espaço
 - A validação feita para um campo do tipo **text** serve também para um **textarea** e para um **password**
 - Validar espaços pode ser feito usando expressões regulares





Validações de formulários

- Validação de campo de texto com preenchimento obrigatório:

```
function validaCampoTexto(id) {  
    var valor = document.getElementById(id).value;  
    //testa se o valor é nulo, vazio ou formado por apenas  
    espaços em branco  
    if ( (valor == null) || (valor == "") || (/^\s+$/ .test(valor)) ) {  
        return false;  
    }  
    return true;  
}
```





Validações de formulários

- Validação de tamanho em campos de texto:
 - É importante validar primeiramente se o campo tem algo preenchido (validação anterior)
 - Pode-se limitar o campo a um tamanho mínimo ou máximo
 - Usa-se o atributo **length** para se checar o tamanho do campo
valor do componente do formulário





Validações de formulários

- Validação de tamanho em campos de texto:

```
function validaCampoTextoTamanho(id, minimo, maximo) {  
    var valor = document.getElementById(id).value;  
    if (!validaCampoTexto(id)) {  
        return false;  
    }  
    if ( (valor.length < minimo) || (valor.length > maximo)) {  
        return false;  
    }  
    return true;  
}
```





Validações de formulários

- Validar para que um campo tenha apenas números:
 - Pode-se validar um campo que deva ser numérico usando-se a função `isNaN` que retorna verdadeiro se um parâmetro não é um número
 - Também é aconselhável validar se o campo contém algum valor





Validações de formulários

- Validar para que um campo tenha apenas números:

```
function validaCampoNumerico(id) {  
    var valor = document.getElementById(id).value;  
    if (isNaN(valor) ) {  
        return false;  
    }  
    return true;  
}
```





Validações de formulários

- Validar se um item foi selecionado numa caixa de seleção ou combo box:
 - Deve-se obter o índice do elemento selecionado através do atributo **selectedIndex**
 - **selectedIndex**: começa do 0 e tem o valor -1 se não houver seleção
 - O índice pode ser nulo se o componente não for do tipo select





Validações de formulários

- Validar se um item foi selecionado numa caixa de seleção ou combo box

```
function validaCampoSelect(id) {  
    var indice = document.getElementById(id).selectedIndex;  
    if ( (indice == null) || (indice < 0) ) {  
        return false;  
    }  
    return true;  
}
```





Validações de formulários

- Validar se uma caixa de checagem (checkbox) está marcada:
 - Deve-se consultar o atributo **checked** do componente

```
function validaCampoCheckbox(id) {  
    var elemento = document.getElementById(id);  
    if (!elemento.checked) {  
        return false;  
    }  
    return true;  
}
```





Validações de formulários

- Validar se pelo menos um botão de radio de um conjunto foi selecionado:

- Os campos radio funcionam em conjunto desde que possuam o mesmo atributo name, portanto não se deve consultar pelo id e sim pelo nome pelo método:

`document.getElementsByName(nome);`

- **getElementsByName(nome)** retorna um array de elementos com o mesmo nome.
- Esse array deve ser percorrido verificando-se no atributo **checked** se pelo menos um dos botões de radio foram marcados





Validações de formulários

- Validar se pelo menos um botão de radio de um conjunto foi selecionado:

```
function validaCamposRadio(nome) {  
    var opcoes = document.getElementsByName(nome);  
    var selecionado = false;  
    for(var i = 0; i < opcoes.length; i++) {  
        if(opcoes[i].checked) {  
            selecionado = true;  
            break;  
        }  
    }  
    if(!selecionado) {  
        return false;  
    }  
    return true;  
}
```





Exercicio

- Cria uma página semelhante à figura abaixo e implemente em JS uma calculadora com as 4 operações fundamentais

Valor 1:

Valor 2:

Operação: + ▼

- +
-
- *
- /

- O valor da caixa select poderá ser obtido da mesma forma que se obtém o valor das caixas de texto
- O resultado do cálculo deve ser exibido com uma função alert
- *Use a função `parseFloat` para converter números reais*



FIM!!!

Duvidas e Questões?

