

# INTRODUÇÃO AO JAVASCRIPT

Prof. Roberson Alves

# APRESENTAÇÃO

**Professor: Roberson Junior Fernandes Alves**

**Formação: Mestrando em Computação Aplicada  
- UEPG**

**Atividade atual: Professor do Curso de Ciência  
da Computação**

**Contatos:**

**E-mail: [robersonjfa@gmail.com](mailto:robersonjfa@gmail.com) e  
[roberson.alves@unoesc.edu.br](mailto:roberson.alves@unoesc.edu.br)**

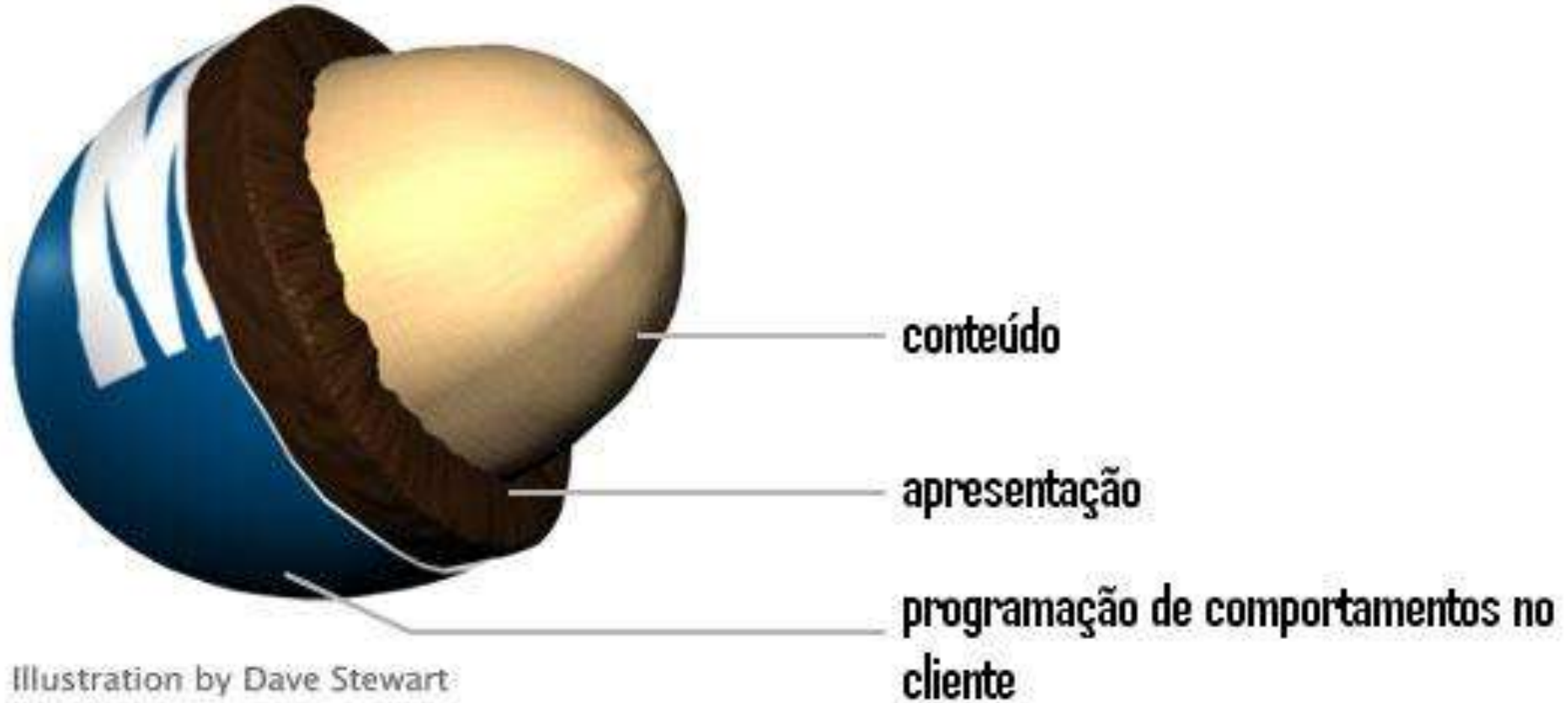
**Skype: [robersonjfa](#) / Twitter: [@robersonjfa](#)**

- ✓ **O que é o JavaScript?**
- ✓ **Características do JavaScript**
- ✓ **Bibliotecas**
- ✓ **Formas de Uso**
- ✓ **Variáveis e Tipos de dados**
- ✓ **Estruturas condicionais e de repetição**
- ✓ **Funções, Objetos e Eventos**
- ✓ **Validação de formulários**

# INTRODUÇÃO

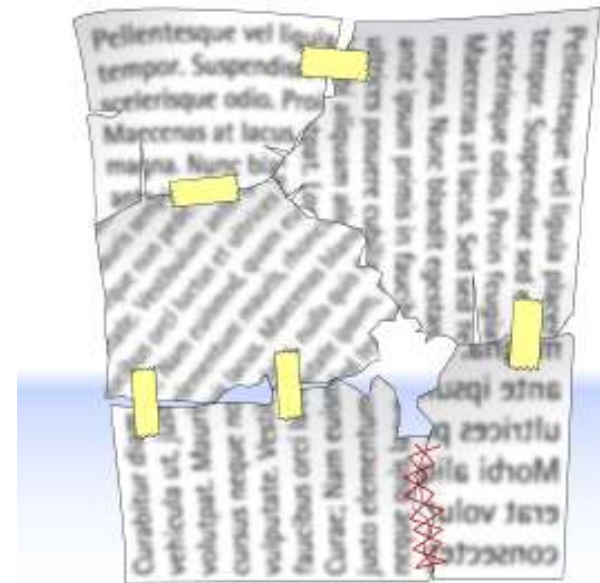
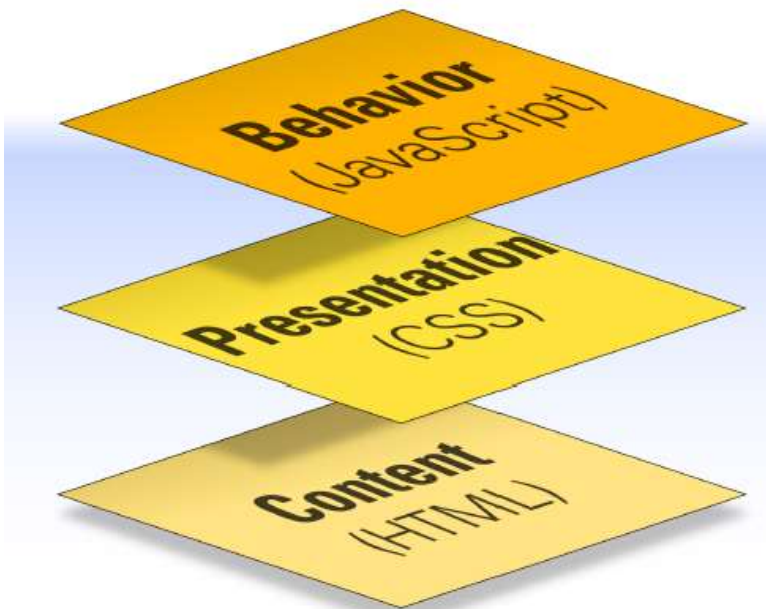
- O código em uma página pode ser concebido em três visões distintas:
  - Estrutura e conteúdo: **HTML**
  - Apresentação: **CSS**
  - Comportamento: **JavaScript**
- Vantagens:
  - Reuso de partes do projeto
  - Modularidade
  - Flexibilidade e facilidade de manutenção
  - Legibilidade
  - ....

# INTRODUÇÃO



# INTRODUÇÃO

- As 3 visões de concepção de uma página nos dão a visão em camadas, em vez de um código “**macarronado**” ou “**remendado**”



# O QUE É JAVASCRIPT?

- Linguagem de Scripting Dinâmica com características OO
- Executada, **principalmente**, no **navegador**;
- Linguagem **+ Popular** do mundo;
- Linguagem **+ Incompreendida**;
- Serve para tudo:
  - Programas completos;
  - Games;
  - Animações;
  - Etc.

- **Desenvolvida por Brendan Eich na Netscape em 1995**
  - Scripting language for Navigator 2
- **Padronizada para compatibilidade com navegadores**
  - ECMAScript Edition 3 (aka JavaScript 1.5)
- **Relacionada ao Java apenas pelo nome**
  - Name was part of a marketing deal
- **Várias implementações disponíveis**
  - Spidermonkey interactive shell interface
  - Rhino: <http://www.mozilla.org/rhino/>



- É uma linguagem poderosa, com sua grande aplicação (não é a única) do lado cliente (browser);
- É uma linguagem de scripts que permite interatividade nas páginas web;
- É incluída na página HTML e interpretada pelo navegador;
- É simples, porém pode-se criar construções complexas e criativas;
- Multiplataforma;
- Padronizada: **ECMAScript.**







# JavaScript é uma linguagem imperativa



**JavaScript NÃO é JAVA**



**Mais uma vez:  
JavaScript NÃO é JAVA**



**Só para deixar claro:  
JavaScript NÃO é JAVA**

# **ALGUMAS COISAS QUE SE PODE FAZER COM JS**

- **Validar entrada de dados em formulários: campos não preenchidos ou preenchidos incorretamente poderão ser verificados;**
- **Realizar operações matemáticas e computação;**
- **Abrir janelas do navegador, trocar informações entre janelas, manipular propriedades como histórico, barra de status, plug-ins, applets e outros objetos;**
- **Interagir com o conteúdo do documento tratando toda a página como uma estrutura de objetos;**
- **Interagir com o usuário através do tratamento de eventos; ...**

# JAVASCRIPT NÃO OBSTRUTIVO

- Que o conteúdo da página deve estar presente e funcional, ainda que se perca em usabilidade, caso o usuário esteja visualizando o documento em um dispositivo (por exemplo, navegador) sem suporte para JavaScript;
- Usar a linguagem com vistas a unicamente incrementar a usabilidade da página;
- Escrever scripts em arquivos externos para serem linkados ao documento e não inserir script na marcação HTML; e
- JavaScript X Acessibilidade.



# ALGUMAS BIBLIOTECAS

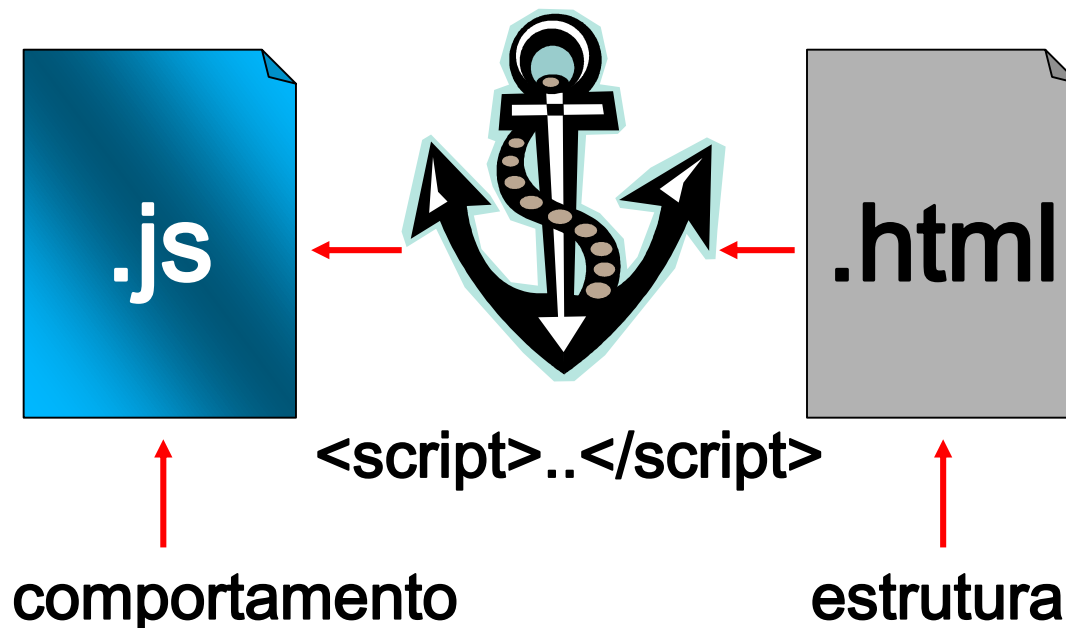


# FORMAS DE USO

- Dentro próprio código HTML:  
`<a href="#" onclick="alert('alô mundo!')">Diga alô</a>`
- Separado em uma tag de script (preferencialmente dentro da tag `<head></head>`):  
`<script type="text/javascript">`  
`alert("alô mundo");`  
`</script>`
- Mais separado ainda dentro de um arquivo “texto” com extensão .js sendo chamado por uma tag script:  
`<script type="text/javascript" src="script.js"></script>`

# DOIS ARQUIVOS SEPARADOS?

- Usaremos dois arquivos texto:
  - Um com HTML com extensão .html
  - Outro com JavaScript com extensão .js
  - Haverá ainda uma tag HTML que “unirá” os arquivos



# ALÔ MUNDO - VERSÃO 1

```
<html>
<head>
  <title>Alô!</title>
  <script type="text/javascript" >
    alert("Alô 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

- 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("alô");  
 alert("mundo");

- **Comentários de uma linha:**  
`alert("teste"); // comentário de uma linha`
- **Comentário de várias linhas:**  
`/* este é um comentário  
de mais de uma linha */`
- **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 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";
```

← concatenação

```
nome = nome + " maria";
```

← concatenação

```
endereco = "rua a, numero " + 3;
```

← concatenação com  
conversão  
numérica  
implícita

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

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

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

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

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

```
var cidades = [ ];
```

```
cidades[0] = "Parnaíba";
```

```
cidades[1] = "Teresina";
```

```
cidades[2] = "LC";
```

```
alert("A capital do Piauí é " + cidades[1]);
```



- **Tamanho de um array: usamos a propriedade length do próprio array**  
`alert(cidades.length);`
- **Último item de um array:**  
`alert(cidades[cidades.length-1]);`

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

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

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

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

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

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

- **Lógico:** tipo que pode ter os valores **true** ou **false**  
**var** aprovado = **true**;  
**alert**(aprovado);

- **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;
```

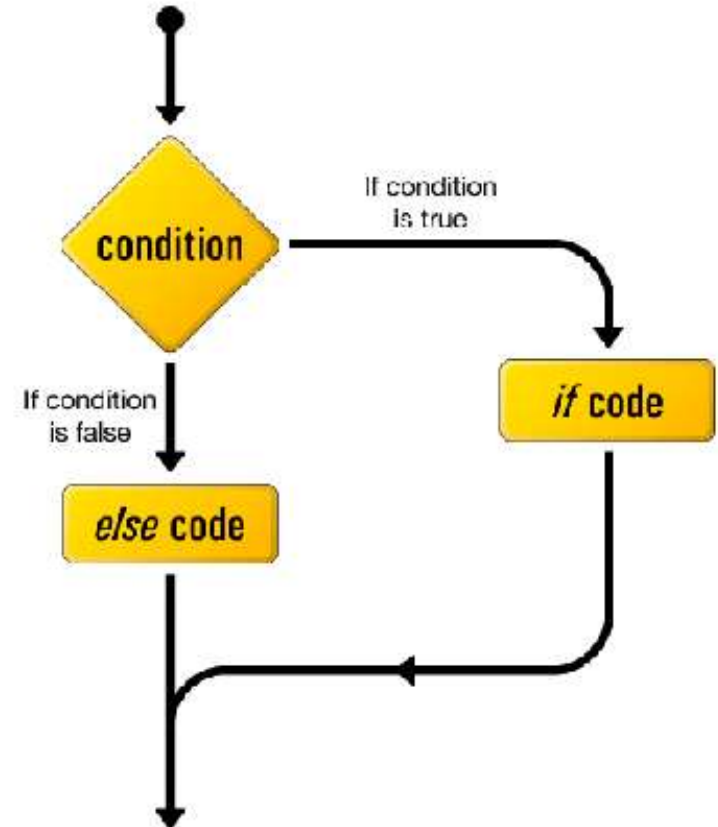
```
alert( typeof b); // undefined
```

- **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

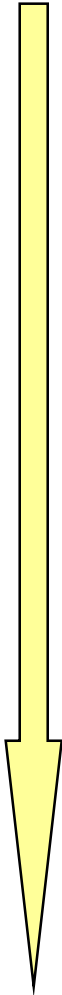
- && : and
- || : or
- ! : not

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

# TABELA DE OPERADORES E PRIORIDADE

Prioridade

+



-

Operador	Descrição
. [] ()	Acesso a propriedades, indexação, chamadas a funções e sub-expressões
++ -- ~ ! new delete typeof	Operadores unários e criação de objetos
* / %	Multiplicação, divisão, módulo
+ -	Adição, subtração, concatenação de strings
<< >> >>>	Deslocamento de bit
< <= > >= instanceof	Menor, menor ou igual, maior, maior ou igual, instanceof
== != === !==	Igualdade, desigualdade, igualdade estrita, e desigualdade estrita
&	AND bit a bit
^	XOR bit a bit
	OR bit a bit
&&	AND lógico
	OR lógico
? :	Operador condicional (ternário)

# 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;
}
```

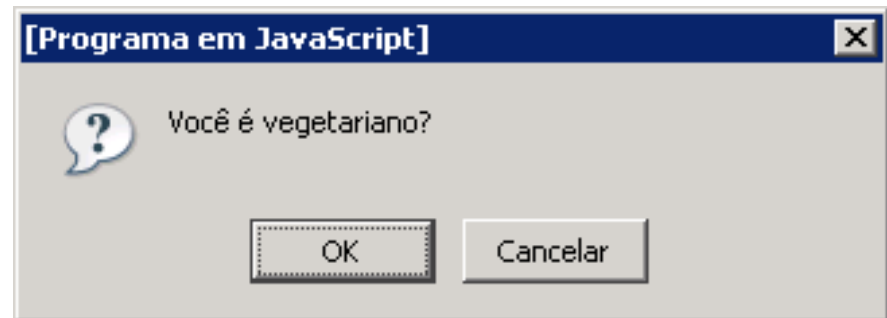
# JANELAS DE DIÁLOGO - CONFIRMAÇÃO

- Nos permite exibir uma janela popup 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");
}
```

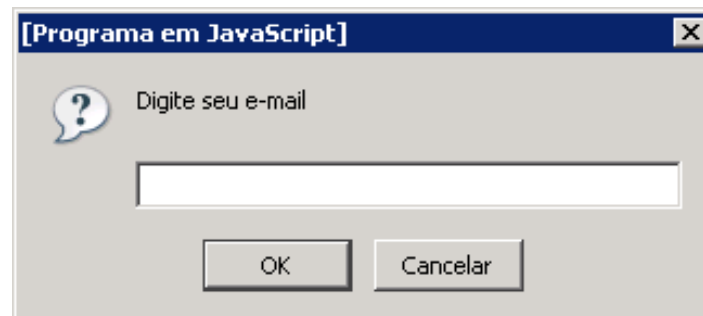
```
else {
    alert("Coma menos gordura");
}
```



# 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/>");  
}
```

# AÇUCARES SINTÁTIXOS

- 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**

- **Elabore scripts usando a função prompt que:**
  - **Leia um valor e imprima os resultados: “É maior que 10” ou “Não é maior que 10” ou ainda “É igual a 10”**
  - **Some dois valores lidos e imprima o resultado**
  - **Leia 2 valores e a operação a ser realizada (+, -, \* ou /) e imprima o resultado (use um switch)**
  - **Leia um nome e um valor n e imprima o nome n vezes usando o laço for**

# 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) ;
```

# TRATAMENTO DE ERROS

## TRY CATCH

- Erros comuns devem ser tratados/evitados;
- Testar para verificar possíveis erros comuns;
- Exceções podem ser tratadas com: **try, catch e finally;**

```
/* Considerando a função do exemplo anterior já declarada */

try {
    lerarr(); //função que pode lançar exceção
} catch (e) {
    // comandos a serem executados em caso de exceção
    window.alert("Oops! No soup for you!");
} finally {
    document.write("ocorreu um erro");
}
```

- Acessar o tipo/nome e mensagem pelas propriedades: **name** e **message**

# 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**.



- 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;
    ...
}
```

- Ex. 1:

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

← alomundo.html

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

← alomundo.js

- Ex. 2:

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

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

← alomundo.html

← alomundo.js

- Ex. 3: Passando parâmetros

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

← saudacao.html

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

← saudacao.js

- 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

- 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;

- **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();" >
...
```



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

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

- **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>

...
```

```
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/>";  
}
```

- **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>
```

```
function validaCampo() {
    var valor = document.getElementById("q").value;

    if ((valor == null) || (valor == "")) {
        alert("Preencha o campo de busca");
        return false;
    }

    return true;
}
```

- **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')"/>
```

...

```
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/>";  
}
```

- **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.

...

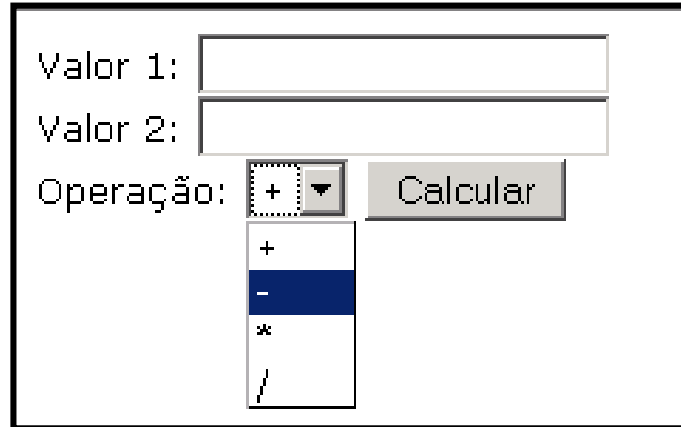
```
<input type="text" name="txtOrigem" id = "txtOrigem"
      onkeydown = "copiaTexto('txtOrigem','txtDestino')"/>
<input type="text" name="txtDestino" id = "txtDestino" />
```

...



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

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



Valor 1:

Valor 2:

Operação: + - \* / Calcular

- 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*

1. **Elabore um formulário HTML que tenha como entrada 3 valores para lados de um triângulo e escreva uma função de nome **tipoTriangulo** que receba 3 parâmetros esses lados de um triângulo e imprima o tipo dele em uma div (equilátero, isósceles ou escaleno).**

**A passagem dos parâmetros deve ser feita de forma simplificada dentro do HTML no evento onclick de um botão ou link da seguinte forma:**

```
<.... onclick = "tipoTriangulo(txtLado1.value, txtLado2.value, txtLado2.value)"...>
```

**2. Deseja-se calcular a conta de consumo de energia elétrica de uma casa. Para isso, elabore um formulário em HTML que leia a quantidade de Kwh consumidos e o valor unitário do Kwh.**

**Escreva uma função em JavaScript que faça o cálculo (valor = quantidade x valor unitário) e, caso a quantidade de Kwh ultrapasse 100, o valor do Kwh deve ser acrescido em 25%. Caso ultrapasse 200, o mesmo valor deve ser acrescido em 50%.**

**Os valores devem ser repassados para uma função em JavaScript conforme o exemplo anterior**

# MODELO DE OBJETOS

## JAVASCRIPT - OBJETOS

- Ao contrário de uma variável, um objeto pode conter diversos valores e de tipos diferentes armazenados nele (atributos);
- Também possuem funções que operem sobre esses valores (métodos);
- Tanto os atributos, quanto os métodos, são chamados de **propriedades** do objeto.

```
var objeto = new Object();
// Quando usamos o construtor Object() criamos um objeto
// genérico
```

# MODELO DE OBJETOS

## JAVASCRIPT - OBJETOS

- **Ex.:**

```
var nave = {
    nome: "coração de ouro",
    propulsao: "Gerador de improbabilidade infinita",
    dono: "Zaphod Bebbblebrox"
}
// Dessa forma, o objeto nave foi criado possuindo os atributos
// nome, propulsão e dono com seus respectivos valores
```

```
function Carro(modelo, marca, ano, motor) {
    this.modelo = modelo;
    this.marca = marca;
    this.ano = ano;
    this.motor = motor;
}
// Depois para instanciar um objeto, basta usar:
var car = new Carro("G800" , "Gurgel" , 1976 , 1.0);

// Agora car já possui todos os atributos com dados:
document.write("Carro: "+car.modelo);
// o comando acima irá imprimir "Carro: G800"
```

# MODELO DE OBJETOS

## JAVASCRIPT - OBJETOS

- **Ex.:**

```
// Uma função fictícia para cálculo de um consumo de combustível
function calc_consumo(distancia) {
    return distancia/(15/this.motor);
}

// Agora atribuímos a função, sem os argumentos, para a
// propriedade consumo. Considerando o objeto já instanciado
// do exemplo anterior
car.consumo = calc_consumo;

// Pronto! já podemos invocá-la fazendo:
var gas = car.consumo(200);
// calculando quanto o carro gastaria de
// combustível em 200 quilômetros
```

# **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 OBJETOS JAVASCRIPT

- ✓ **Objetos de navegação**
  - ✓ **criando uma janela**
    - ✓ **`window.open(URL, Nome, "height=400,width=600")`**

# 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)

- DOM é um padrão W3C (World Wide Web Consortium)
- Define um padrão para acesso a elementos de documentos como HTML e XML
- **Document Object Model** (DOM) é uma plataforma e uma interface de linguagem-neutra que possibilita aos programas e scripts acessarem e atualizarem dinamicamente o conteúdo e o estilo de um documento.

- **Separado em 3 partes / níveis:**
  - **Core DOM** – modelo padrão para qualquer estrutura de documento
  - **XML DOM** – modelo padrão para documentos XML
  - **HTML DOM** - modelo padrão para documentos HTML
- **DOM define objetos e propriedades para todos elementos do documentos e métodos (interfaces) para acesso a eles**

- **DOM fornece a representação estrutural de documentos HTML e XML, definindo a forma como a estrutura que pode ser acessada por programas e scripts, possibilitando a modificação na estrutura do documento, do estilo e do conteúdo**
- **HTML DOM é o padrão de como acessar, alterar, acrescentar e deletar elementos HTML.**

- De acordo com a DOM tudo em documento HTML são nodos
  - O documento todo é um nodo
  - Tags html são element nodes
  - Texto dentro de tags são text nodes
  - Atributos são attribute nodes
  - Comentários são comment nodes

# DOM - NODOS

```
<html>
  <head>
    <title>Titulo</title>
  </head>
  <body>
    <h1>Texto</h1>
    <p>Hello world!</p>
  </body>
</html>
```

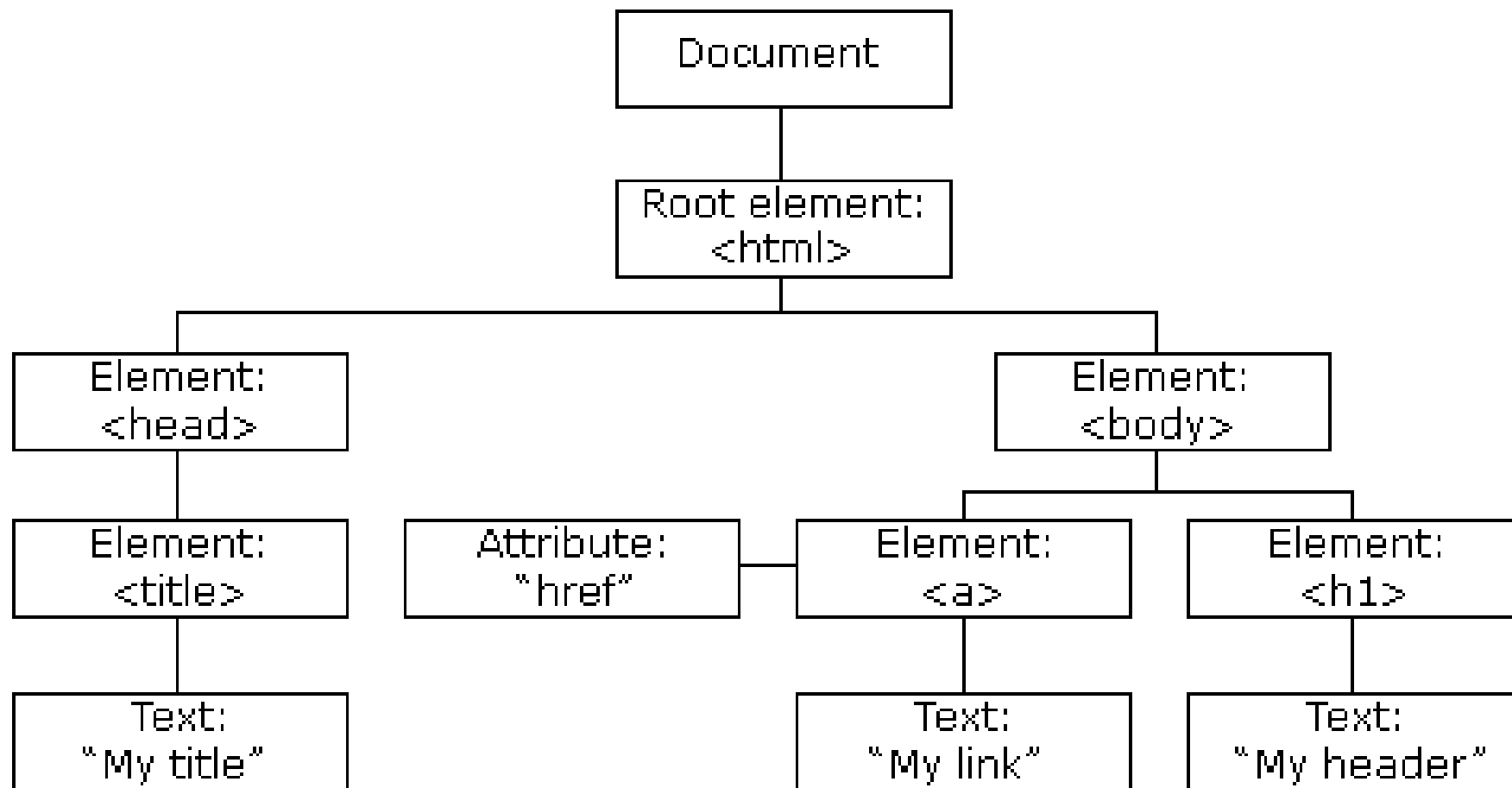
- **<html> → root node**
- **<html> têm dois filhos <head> <body>**
- **Erro comum é esperar que um element node contém texto**
  - **O Texto de um element node é armazenado em um text node**
  - **“Titulo” não é o o valor do elemento <title>**



# DOM – ÁRVORE DE NODOS

- **DOM Representa o documento através de uma árvore de nós (*tree of node*) em que cada objeto possui sua propriedade e método**

# DOM – ÁRVORE DE NODOS



## Node-tree

# DOM – ÁRVORE DE NODOS

- Hierarquia de nodos
  - Nodo top é chamado de nodo raiz (root)
  - Cada nodo, exceto root, têm exatamente um pai
  - Um nodo pode ter qualquer número de filhos
  - Um nodo leaf é um nodo sem filhos
  - Irmãos (siblings) são nodos com o mesmo pai

```
<html>
  <head>
    <title>Titulo</title>
  </head>
  <body>
    <h1>Texto </h1>
    <p>Hello world!</p>
  </body>
</html>
```

- 3 maneiras
  - 1. Usando getElementById()
  - 2. Usando getElementsByTagName()
  - 3. Navegando pela árvore de nodos usando os relacionamentos entre nodos

```
<html>
  <body>
    <p id="intro">W3Schools example</p>
    <div id="main">
      <p id="main1">The DOM is very useful</p>
      <p id="main2">This example demonstrates
        <b>node relationships</b>
      </p>
    </div>
  </body>
</html>
```

```
var x=document.getElementById("intro");
var text=x.firstChild.nodeValue;
```

- **Duas propriedades que permitem acesso as tags**
  - **document.documentElement**
    - Retorna o nodo root do documento e existe em todos documentos XML e HTML
  - **document.body**
    - Acesso direto a tag <body>

# DOM – MÉTODOS E PROPRIEDADES

- DOM modelo o HTML como um conjunto de objetos (cada nodo é um objeto)
- Estes nodos pode ser acessados por JavaScript ou qualquer outra linguagem de programação
- A API é definida pelo conjunto de métodos e propriedades
- Propriedades: referenciado como algo que é (ex. nodename é p)
- Métodos: referenciado como algo que é feito (ex. deletar p)

# DOM – MÉTODOS E PROPRIEDADES

- **Propriedades:**
  - **Sendo x um objeto (elemento HTML)**
    - **x.innerHTML** – o conteúdo (texto) de x
    - **x.nodeName** – o nome de x
    - **x.nodeValue** – o valor de x
    - **x.parentNode** – o pai de x
    - **x.childNodes** – os filhos de x
    - **x.attributes** – os nodos atributos de x

# DOM – MÉTODOS E PROPRIEDADES

- **Métodos:**
  - Sendo **x** um objeto (elemento HTML)
    - **x.getElementById(*id*)** – obtém o elemento especificado pela ID
    - **x.getElementsByTagName(*name*)** – obtém todos os elementos especificados pelo nome
    - **x.appendChild(*node*)** – insere um elemento filho no node **x**
    - **x.removeChild(*node*)** – remove um elemento filho do node **x**



- **getElementById**
  - Provavelmente o mais utilizado do DOM
  - Retorna o elemento cujo atributo ID foi especificado

- Criando um campo em um formulário

```
<input type="text" id="nomeContato" name="nomeContato" ...
```

- Pode ser acessado via javascript através de

```
document.nomeForm.nomeContato.value
```

- Se for usado o método pode ser acessado somente pela ID

```
document.getElementById('nomeContato').value
```

- getElementById

```
<script type="text/javascript">
  <!--
  function showNameJS() {
    var name = document.frm.nomeContato.value;
    alert(name);
  }
  function showNameDOM() {
    var name =
    document.getElementById('nomeContato').value;
    alert(name);
  }
  //-->
</script>
```

- **getElementsByTagName**
  - Retorna diversos objetos com a tag especificada no método.
  - Pode retornar vários objetos → Acessar via índice  
`x=document.getElementsByTagName("p");`  
`y = x[1]`

- **getElementsByTagName**

- Exemplo de script que altera a propriedade textDecoration de tags <a>

```
<script type="text/javascript">
  <!--
  function changeStyleLink() {
    var links = document.getElementsByTagName('a');
    for(var x = 1; x < links.length; x++) {
      var _link = links[x];
      _link.style.textDecoration = 'underline';
    }
  }
  //-->
</script>
```

- **style**
  - Altera o estilo de um objeto
  - Exemplo de função usada para alterar a propriedade display de um objeto ocultando-o

```
function showHide(id) {
    var obj = document.getElementById(id);
    if(obj.style.display == "") {
        obj.style.display = "none";
    } else {
        obj.style.display = "";
    }
}
```

- **className**
  - Semelhante ao **style** tem a função de alterar o estilo de um objeto definido por uma classe CSS
  - A vantagem do método **className** é que podemos alterar as propriedades do estilo na folha de estilo vinculada à página, ou seja, não precisamos alterar o script quando desejarmos alterar o estilo e sim as propriedades da classe CSS.

- **className**

```
<p>
  <a href="javascript:;" onclick="showHide('box');">Ocultar / Exibir o
box.</a>
</p>
<div id="box"> <h1>Teste.</p> </div>
```

– Script usado para ocultar o objeto

```
function showHide(id) {
  var obj = document.getElementById(id);
  if(obj.className == "") {
    obj.className = "ocultaObj";
  } else {
    obj.className = "";
  }
}
```

```
.ocultaObj {
  display: none;
}
```

- **innerHTML**
  - Para alterarmos ou inserirmos conteúdo ou uma marcação HTML em um objeto, utilizamos o método **innerHTML**



- **innerHTML**

```
<p>
  <a href="javascript:;" onclick="javascript:alterConteudo(); return
false;">Alterar a marcação HTML .</a>
</p>
<div id="box"> <h1>Teste.</p> </div>
```

- **Script usado**

```
script type="text/javascript">
<!--
function alterConteudo() {
  var obj = document.getElementById("box");
  obj.innerHTML = "<h1>Esta é a nova marcação HTML desta tag.</h1>";
}
//-->
</script>
```

# DOM – PROPRIEDADES

- **Informações dos nodos**
- **Três propriedades dos nodos importantes:**
  - **nodeName**
  - **nodeValue**
  - **nodeType**

- **nodeName**
  - Especifica o nome do nodo
  - nodeName é somente leitura
  - nodeName de um elemento é o mesmo que sua tag name
  - nodeName de um atributo é o nome do atributo
  - nodeName de um texto é sempre #text
  - nodeName de um documento é sempre #document
- **Nota: nodeName sempre contém a tag de um elemento HTML em maiúsculo**

- **nodeValue**
  - Especifica o valor de um nodo
  - **nodeValue** de um **node element** é indefinido
  - **nodeValue** de um elemento texto é o próprio texto
  - **nodeValue** de um atributo é o valor do atributo

```
x=document.getElementById("intro").firstChild;  
txt=x.nodeValue;
```

- **nodeType**
  - Retorna o tipo do nodo
  - Os mais importantes são

ElementType	NodeType
Element	1
Attribute	2
Text	3
Comment	4
Document	5

# JAVASCRIPT: 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

- 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;
}
```

- **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

- 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;
}
```

- **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**

- Validar para que um campo tenha apenas números:

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

- 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

- 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;
}
```



- 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;
}
```

- 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

- 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;
}
```

- **Nas atividades seguintes:**
  - Use uma página HTML e um arquivo de scripts
  - Use o evento **onsubmit** do formulário e uma função de validação que retorne **true** ou **false**
  - Utilize uma página qualquer como **action** do formulário.

- Copie o valor de um campo texto para outro caso o campo de origem não esteja vazio. Use o evento onblur do campo de origem
- Valide um campo senha de acordo com seu tamanho:
  - < 3: segurança fraca
  - Entre 3 e 5: segurança média
  - >= 6: segurança forte
- Valide se dois campos do tipo **password** são idênticos
- Valide 3 campos texto que representem dia, mês e ano:
  - Dia: entre 1 e 31
  - Mês: entre 1 e 12
  - Ano: > 1949