

# Introdução à Programação para WEB

Introdução ao JavaScript

Introdução, comandos, elements, functions e eventos

Adaptado do Slide professor Ely Miranda

1

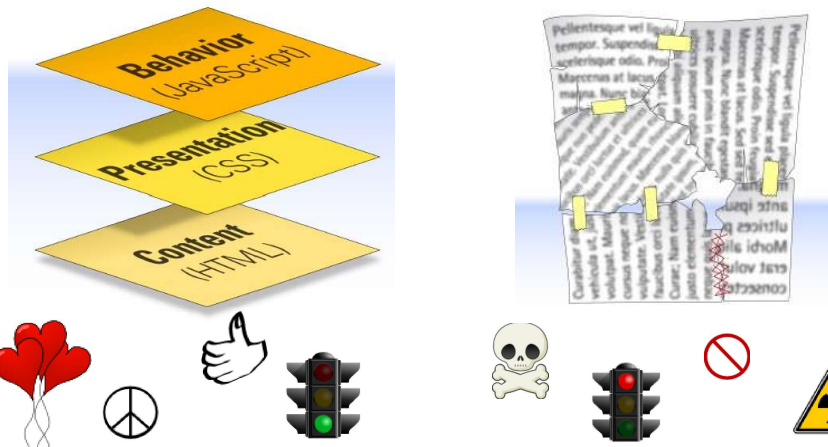
## Introdução

- 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
  - ....

2

## Introdução

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



## Características

- 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
- JavaScript não é Java. São linguagens com aplicações e recursos totalmente distintos

### **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 com 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

5

### **Algumas bibliotecas**

- **Prototype:** <http://www.prototypejs.org/>
- **script.aculo.us:** <http://script.aculo.us/>
- **Yahoo! User Interface Library (YUI):**  
<http://developer.yahoo.com/yui/>
- **Dojo:** <http://dojotoolkit.org/>
- **jQuery:** <http://jquery.com/>
- **MooTools:** <http://mootools.net/>

6

## 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>
```

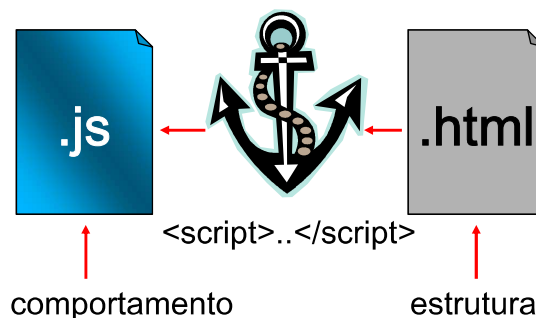
- Mais separado ainda dentro de um arquivo “texto” com extensão .js sendo chamado por uma tag script:

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

7

## 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



8

## Alô mundo - versão 1

```
<html>
<head>
  <title>Alo!</title>
  <script type="text/javascript" >
    alert("Alo mundo!");
  </script>
</head>
<body>
  ...
</body>
</html>
```

9

## Alô mundo – versão 2

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

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

alomundo.html

alomundo.js

10

## 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");

11

## 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.

12

## 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);

13

## 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

14

## 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 cidades = [ ];
```

```
cidades[0] = "Parnaíba";
```

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

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

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

15

## Alguns tipos

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

```
alert(cidades.lenght);
```

- Último item de um array:

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

16



## 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["jose"] = 48;  
idades["maria"] = 20;  
alert("Minha idade é: " + idades["ely"]);
```

17

## Alguns tipos

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

```
var aprovado = true;  
alert(aprovado);
```

18

## 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
```

19

## Operador de tipos

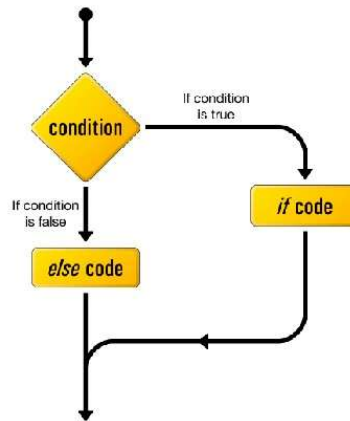
- Utilizando typeof podemos ter os seguintes resultados:
  - undefined: se o tipo for indefinido.
  - boolean: se o tipo for lógico
  - number: se for um tipo numérico (inteiro ou ponto flutuante)
  - string: se for uma string
  - object: se for uma referência de tipos (objeto) ou tipo nulo

20

## 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

21

## 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");
}
```

22

## Estruturas de decisão – if e else

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

23

## 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;  
}  

```

24

## Estruturas de decisão – Switch

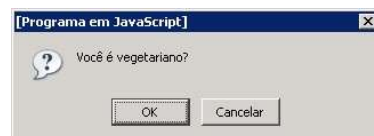
```
var idade = 20;
switch (idade) {
  case 29:
    alert("Você está no auge.");
    break;
  case (40) :
    alert("A vida começa aqui.");
    break;
  case (60) :
    alert("Iniciando a melhor idade.");
    break;
  default:
    alert("A vida merece ser vivida, não importa a idade.");
    break;
}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

25

## 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");
}
```

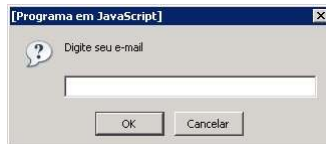


26

## 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.");
```



27

## 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 (2,10,16...)
- **parseFloat(valor)**: converte uma string para um valor real/ponto flutuante

28

## 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 );
```

29

## 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/>");  
}
```

30

## 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

31

## Exercícios

- 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**

32



## 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;  
}
```

33

## 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);
```

34

## 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**

35

## 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;  
    ...  
}
```

36

## Funções

- Ex. 1:

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

← alomundo.html

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

← alomundo.js

37

## Funções

- Ex. 2:

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

← alomundo.html

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

← alomundo.js

38

## Funções

- 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

39

## 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>  
...
```

40

## 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

41

## 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;

42

## 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();" >  
...
```

43

## Eventos

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

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

44

## 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>  
  
...
```

45

## 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/>";  
}
```

46

## 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>
```

47

## Eventos

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

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

48



## 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')"/>
```

...

49

## 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/>";  
}
```

50

## 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" />
```

...

51

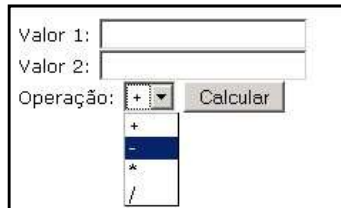
## Eventos

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

52

## Prática

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



- 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*

53

## Prática

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, txtLado3.value)" ...>
```

54

## Prática

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 para uma função em JavaScript conforme o exemplo anterior

55

## 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.

56

## 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

57

## 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

58

## 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

59

## 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;  
}
```

60

## 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

61

## 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;  
}
```

62

## 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

63

## 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;  
}
```

64



## 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

65

## 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;  
}
```

66

## 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;  
}
```

67

## 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

68

## 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;  
}
```

69

## Prática

- 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.

70

## Prática

- Copie o valor de um campo texto para outro caso o campo de origem não esteja vazio. Use o evento on blur 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

71