PROGRAMAÇÃO DE SCRIPTS

JAVASCRIPT

RESUMO DAS AULAS

Prof. Dr. Celso Gallão

JavaScript

Este material é apenas um resumo para acompanhamento dinâmico das aulas ministradas, sendo necessário o acompanhamento nas aulas e as orientações do professor.

PARTE 01 - Introdução

1 - Visão Geral

A linguagem de programação **JavaScript** foi desenvolvida pela **Netscape** e é utilizada para aumentar a interatividade das páginas *web*. O **VBScript** foi a resposta da **Microsoft** para o **JavaScript**. O **HTML** define o conteúdo da página, o **CSS** define o *layout* da página e o **JavaScript** permite que seja programado o comportamento da página *web*.

JavaScript não deve ser confundido com a linguagem de programação **Java**, pois esta é uma linguagem desenvolvida pela **Sun Microsystem**, repleta de recursos que permitem criar aplicações independentes, com recursos de uma linguagem destinada à criação de aplicações comerciais completas e ainda controlar dispositivos eletrônicos domésticos, como a linguagem **C++** por exemplo. **Java** é especializado na criação de pequenos programas chamados **Applets**, distribuídos através da internet.

1.1 - Características do JavaScript

É uma linguagem que, normalmente, se aloja dentro de uma página HTML, ou seja, o **JavaScript** é um programa incluído em uma página HTML. Para tanto, deve-se decralar na *tag* **`<SCRIPT>...</SCRIPT>`** que a linguagem será JavaScript. O código escrito dentro da *tag* **`<SCRIPT>`** não é renderizada para visualização na tela. A *tag* **`<SCRIPT>`** pode ser escrita na seção **`<HEAD>`** do HTML, normalmente para declarar funções de autoria do desenvolvedor, mas também pode estar em qualquer parte da seção **`<BODY>`**.

JavaScript é uma linguagem interpretada, baseada em objetos, o que significa que podemos utilizar objetos predefinidos ou criar novos objetos. Assim, **JavaScript** permite a criação de páginas HTML dinâmicas não compiladas, sem listagens de erros para correção.

JavaScript é *keysensitive*, ou seja, sensível a diferença entre letras maiúsculas e minúsculas. Isso significa que ***myName*** é diferente de ***MyName***, que é diferente de ***myname*** ou de ***Myname***. O código-fonte **JavaScript** deve ser digitado em letras minúsculas, mas há comandos que possuem apenas uma letra em maiúscula, como por exemplo o evento **`onClick()`**.

1.2 - O que JavaScript é Capaz de Fazer

Ele deixa as páginas *web* mais interativas, proporcionando uma experiência de navegação melhor e mais interessante aos visitantes. Por exemplo, botões dinâmicos, verificar e validar formulários HTML, permitir que o visitante altere as propriedades das páginas como cores e fontes, abrir novas páginas e janelas conforme configurações específicas, exibir mensagens personalizadas, gerar calendários, relógios e documentos com timbre de hora, testar a presença de *plugins* do navegador ou direcionar o visitante para uma outra página, entre muitas outras aplicações.

1.3 - Objetos

Quando um documento é carregado no navegador, ele cria vários objetos JavaScript com propriedades e valores específicos. Cada objeto armazena várias informações que podem ser acessadas ou alteradas pelo desenvolvedor. A seguir, estão listados os principais objetos que compõem uma página HTML em uma janela exibida no navegador:

a) navigator: possui propriedades para o **nome** e a **versão** do navegador.

b) window: possui propriedades que se aplicam à janela como um todo. É o objeto de **maior** nível hierárquico.

c) document: possui informações sobre a **página** HTML como um todo.

d) form: possui informações sobre os **formulários** da página atual.

e) history: possui uma lista de todos os **sites visitados** na sessão de uso atual.

f) location: possui informações sobre o local da página, como o **protocolo** utilizado e seu **domínio**, por exemplo.

g) screen: possui informações sobre a tela do visitante. Não há um padrão público que se aplique ao objeto da tela, mas todos os navegadores principais o suportam.

1.4 - Propriedades

Um **objeto** possui **características** próprias dentro de um grupo, algumas são comuns a muitos **objetos**, outras são específicas de cada um. Essas características chamamos de **propriedades**, e podem ser vistas como **variáveis** que armazenam informações sobre um determinado **objeto**. Portanto, as **propriedades** podem mudar os **objetos**. Sua sintaxe é muito simples:

```
objeto.propriedade = valor;
```

A seguir, 3 exemplos que definem algumas propriedades da página *web*, respectivamente, o título, a cor do fundo e a cor das fontes:

```
<SCRIPT language="javascript">
   document.title   = "Exemplo 1";
   document.bgColor = "lightblue";
   document.fgColor = "darkblue";
</SCRIPT>
```

1.5 - Métodos

Um **objeto** possui rotinas especiais predefinidas que realizam operações relacionadas ao próprio **objeto**. São as **ações** possíveis de serem realizadas pelos **objetos**. Essas ações chamamos de **métodos** e normalmente são utilizados para alterar valores de propriedades ou executar tarefas específicas.

Os **métodos** estão relacionados aos **objetos** e já fazem parte da linguagem **JavaScript**, ou seja, não são criados pelo desenvolvedor web. Sua sintaxe também é bem simples: os parênteses indicam que estamos fazendo referência a **métodos** e não a **propriedades**:

```
objeto.método("argumento");
```

A seguir, 2 exemplos que definem ações realizadas, respectivamente, um texto que será exibido na tela e uma caixa de alerta com uma mensagem:

```
<SCRIPT language="javascript">
   document.write("Teste de JavaScript");
   window.alert("Operação Inválida !");
</SCRIPT>
```

1.6 - Eventos

São **ações** que ocorrem na página *web* devido à interação ocorrida com o usuário, ou seja, são **ações** que o usuário executa quando visita uma página *web*. Por exemplo, enviar um formulário HTML ou mover o ponteiro do mouse sobre um **objeto**.

Uma linguagem baseada em **objetos** possibilita reagir a uma **ação** do usuário, executando os chamados **Tratadores de Eventos**. Os **eventos** são os principais recursos da linguagem **JavaScript** para a validação de campos e alteração dinâmica da página *web*. A seguir temos a tabela com alguns **Tratadores de Eventos**, pois a lista completa é bem extensa:

TRATADORES DE EVENTOS DE MOUSE	
`onClick()`	O usuário clicou sobre o objeto.
`onDblClick()`	O usuário deu duplo-click sobre o objeto.
`onMouseOver()`	O cursor é deslocado para sobre o objeto.
`onMouseOut()`	O cursor é deslocado para fora do objeto.
`onMouseDown()`	O usuário pressiona o mouse.
`onMouseUp()`	O usuário libera o mouse.
`onMouseMove()`	O mouse é movimentado pelo objeto.
`onContextMenu()`	O usuário clicou no botão direito do mouse.

TRATADORES DE EVENTOS DE TECLADO	
`onKeyDown()`	O usuário está pressionando uma tecla.
`onKeyUp()`	O usuário liberou uma tecla.
`onKeyPress()`	O usuário pressionou uma tecla.

TRATADORES DE EVENTOS DE OBJETO	
`onError()`	O *script* encontrou um erro na página.
`onAbort()`	O carregamento de um recurso foi interrompido.
`onLoad()`	O objeto concluiu o carregamento.
`onUnload()`	O objeto foi descarregado.
`onResize()`	O objeto foi redimensionado.
`onScroll()`	A barra de rolagem foi acionada.

TRATADORES DE EVENTOS DE FORMULÁRIO	
`onBlur()`	O objeto perdeu o foco.
`onFocus()`	O objeto recebeu o foco.
`onChange()`	O estado do objeto foi alterado.
`onSelect()`	O usuário selecionou o texto do objeto.
`onSubmit()`	O formulário foi submetido.
`onReset()`	O formulário foi reiniciado com os valores padrão.

TRATADORES DE EVENTOS DE TOQUE	
`onTouchCancel()`	O toque é interrompido.
`onTouchEnd()`	O dedo do usuário é retirado da tela.
`onTouchMove()`	O dedo do usuário é arrastado através tela.
`onTouchStart()`	O dedo do usuário é colocado na tela.

Ao escrever um *script*, não é preciso prever todos os eventos possíveis, apenas aqueles que você deseja que algo especial ocorra. A sintaxe também é bem simples: Os parênteses indicam que estamos fazendo referência a **eventos** e não a **propriedades**:

objeto.evento("argumento");

Por exemplo, o objeto HTML `buttom` reage ao **evento** `onClick()`, que ocorre quando o usuário dá um clique com o ponteiro do mouse sobre o objeto. Neste exemplo, o argumento é uma caixa de alerta com uma mensagem:

```
<BODY>
  <INPUT TYPE="buttom" value="TESTE" name="BT1"
  onClick="alert('Testando evento!')">
</BODY>
```

1.7 - Funções

São rotinas independentes escritas pelo **desenvolvedor**, que executam uma tarefa específica. Elas se diferenciam dos **métodos** pois não são associadas a um **objeto** além de serem geradas pelo desenvolvedor. Existem **funções** genéricas em **JavaScript**, mas o desenvolvedor pode criar suas próprias **funções**.

Toda **função** deve receber um **nome** e pode ser acessada por outras partes do *script* tantas vezes quantas forem necessárias, aliás, essa é a grande vantagem da utilização das **funções**, pois escreve-se o *script* apenas uma vez e o acessamos sempre que necessário.

Aqui, temos apenas uma introdução sobre **funções**. Na **Parte 5** falaremos muito mais sobre este assunto.

a) Criar uma Nova Função:

Declara-se `function` seguido do **nome** escolhido para a função. Sempre colocam-se parênteses após o nome da função. Os códigos da função devem ser escritos dentro de chaves:

```
<SCRIPT>
   function cheguei() {
       window.alert("Querida, cheguei !!!");
   }
</SCRIPT>
```

b) Executar uma Função:

Normalmente utiliza-se um **tratador de eventos** para acessar uma função existente. A função acima pode ser chamada, por exemplo, pelo acionamento do botão HTML descrito a seguir:

```
<BODY>
   <INPUT TYPE="buttom" value="Clique aqui"
   name="btCheguei" onClick="cheguei()">
</BODY>
```

c) Enviar Parâmetros para Funções:

Em alguns casos, há parâmetros declarados dentro dos parênteses do nome da função. Frequentemente, é preciso fornecer dados de entrada para que a função processe e retorne uma saída. Para tanto, ao criar a função, deve-se declarar os nomes dos parâmetros dentro dos parênteses do nome da função, separados por vírgulas se for o caso:

```
<SCRIPT>
   function ola(mens) {
       window.alert(mens);
   }
</SCRIPT>
```

Para enviar os parâmetros para a função, deve-se inserir os valores de entrada dentro dos parênteses do nome da chamada da função:

```
<FORM>
  <INPUT type="button" name="bt1"
  value="Mensagem 1" onClick="ola('Funciona!')">
<BR>
  <INPUT type="button" name="bt2"
  value="Mensagem 2" onClick="ola('Alo mundo!')">
<BR>
  <INPUT type="button" name="bt3"
  value="Mensagem 3"onClick="ola('Sou o cara!')">
</FORM>
```

d) Funções que Retornam Valores

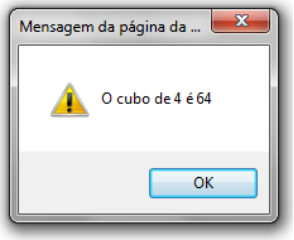
Em alguns casos, há a necessidade de se retornar o resultado de um cálculo para o ponto aonde a função foi chamada, geralmente fora da área de *script*. O comando **`return`** é utilizado para isso:

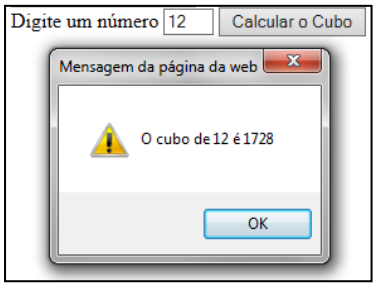
```
<SCRIPT>
   function cubo(x) {
       return x*x*x;
   }
</SCRIPT>
```

Ao enviar os parâmetros para a função, **`return`** irá devolver o resultado da função para o ponto onde a função foi chamada:

```
<FORM>
<INPUT type="button" name="bt1" value="Calcular
   o Cubo de 4" onClick="alert('O cubo de 4 é '
   + cubo(4))">
<BR>
   Digite um número
<INPUT type="text" size=2 name="num">
<INPUT type="button" name="bt2" value="Calcular
   o Cubo" onClick="alert('O cubo de ' +
   num.value + ' é ' + cubo(num.value))">
</FORM>
```

No exemplo acima, quando o usuário clicar sobre o botão chamado **`bt1`**, uma caixa de alerta irá ser exibida, conforme ao lado. Para tanto, a função **`cubo()`** foi chamada tendo como parâmetro o valor **`4`**.Ainda no exemplo acima, o usuário pode digitar um número qualquer na caixa de texto e em seguida sobre o botão chamado **`bt2`**. No exemplo, foi digitado o número 12. Neste caso, a caixa de alerta será exibida conforme abaixo. Para tanto, a função **`cubo()`** foi chamada tendo como parâmetro **`num.value`**, ou seja, o conteúdo digitado na caixa de texto chamada **`num`**:





e) Funções que Não Retornam Valores

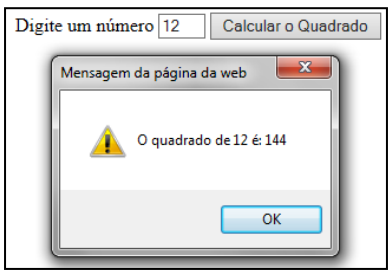
Em outros casos, o resultado de um cálculo de uma função pode ser calculado e utilizado dentro da própria área de *script*, sem utilizar o comando **`return`**:

```
<SCRIPT>
   function quadrado(x)
   {
   y = x*x;
   alert('O quadrado de ' + x + ' é: ' + y)
   }
</SCRIPT>
```

Assim, ao enviar os parâmetros, a função é executada dentro da área de *script*, sem retornar resultado algum. O exemplo abaixo mostra a função **`quadrado()`** sendo chamada com o parâmetro **`num2.value`**, que é o valor digitado na caixa de texto chamada **`num2`**.

```
<FORM>
   Digite um número
<INPUT type="text" size=2 name="num2">
<INPUT type="button" name="bt3" value="Calcular
   o Quadrado"
   onClick="alert(quadrado(num2.value))">
</FORM>
```

O resultado da função **`quadrado()`**, após a digitação do valor **`12`** na caixa de texto, é mostrado na figura abaixo:



PARTE 02 - Elementos da Linguagem

2 - Variáveis

Armazenam temporariamente um conteúdo qualquer, que pode ser um texto, um valor, uma data, etc. Pode receber seu conteúdo através de uma simples atribuição com o operador igual (**=**) ou através do resultado de uma expressão ou função.

O nome de uma **variável** pode começar com uma letra ou sublinhado, seguido de números ou letras, mas sem espaços. O JavaScript é *keysensitive*, portanto, pode-se utilizar a combinação de letras maiúsculas e minúsculas para criar nomes de varáveis.

A tabela de valores abaixo, descreve os tipos de **variáveis** do JavaScript:

TIPOS DE VALORES	
`number`	qualquer valor numérico.
`string`	caracteres, entre aspas.
`boolean`	verdadeiro ou falso, `1` ou `0`, `true` ou `false`.
`null`	vazio e sem significado.
`object`	qualquer valor associado com o objeto.
`function`	valor retornado por uma função.

Para criar uma variável JavaScript pode-se apenas declarar o seu nome e seu respectivo valor. Não é preciso declarar o tipo da variável, pois o JavaScript "perceberá" automaticamente através de seu conteúdo:

```
<SCRIPT>
  nome          = "João da Silva";
  dataHoje      = date();
  total05       = 20*4;
  _codigo2      = "XZP";
</SCRIPT>
```

Entretanto, é importante declarar uma variável JavaScript utilizando o comando **`var`**, seguido do nome e do valor inicial da variável. Isso é para distinguir variáveis locais de globais. Sem declarar **`var`** todas as variáveis serão globais, ou seja, **`var`** define uma variável **local**.

```
<SCRIPT>
  var nome      = "João da Silva";
  var dataHoje  = date();
  var total05   = 20*4;
  var _codigo2  = "XZP";
</SCRIPT>
```

3 - Literais

Representação de **números** ou ***strings***. São informações fixas que, ao contrário das variáveis, não podem ser alteradas. São criadas durante a execução do programa. Os literais podem ser de vários tipos:

a) **`integer`**:

Números positivos, negativos ou fracionários:

```
<SCRIPT>
  var a = 300;
  var b = 0.25;
  var c = -32;
</SCRIPT>
```

b) **`floating point`**:

Números de ponto flutuante, conhecidos também como de notação científica:

```
<SCRIPT>
  var d = 2.34e4;       //(2.34 x 10000)
</SCRIPT>
```

c) **`boolean`**:

Os literais booleanos podem ser **`True`** ou **`False`**:

```
<SCRIPT>
  var e = true;
</SCRIPT>
```

d) **`string`**:

Cadeia de caracteres envolvidos por aspas ou apóstrofos:

```
<SCRIPT>
  var nome_1 = "João da Silva";
  var nome_2 = 'José Amado';
  var nome_3 = "";
  var cep    = "09700-100";
</SCRIPT>
```

e) **Caracteres Especiais**:

Dentro de **`strings`** podem ser especificados caracteres especiais, para formatar mensagens em caixas de diálogos ou *e-mails*, por exemplo:

CARACTERES ESPECIAIS	
`\n`	insere uma quebra de linha.
`\t`	insere um caracter de tabulação.
`\r`	insere um retorno de carro, quebra de linha.
`\a`	insere um *beep* (sinal sonoro).
`\f`	insere um avanço de página ou formulário.
`\b`	insere um *backspace* (caracter em branco).
`\'`	exibe um apóstrofo (aspas simples).
`\"`	exibe aspas.
`\\`	exibe uma barra invertida.

4 - Expressões e Operadores

Uma **expressão** é uma combinação de **variáveis**, **literais**, **métodos**, **funções** e **operadores** que retornam um resultado. É usada para atribuir um valor a uma **variável** ou para ser testada e atribuir uma **ação**. A seguir, os diversos tipos de **operadores** que podem ser usados em **expressões**:

4.1 - Operadores Comuns

São os operadores comuns, desde a aritmética simples de somar e subtrair, até o resultado do módulo que é o resto de uma divisão:

OPERADORES COMUNS	
`x + y`	soma **`x`** com **`y`**.
`x - y`	subtrai **`y`** de **`x`**.
`x * y`	multiplica **`x`** e **`y`**.
`x / y`	divide **`x`** por **`y`**.
`x % y`	módulo de **`x`** e **`y`** (resto da divisão de **`x`** por **`y`**).

4.2 - Operadores Incrementais

São operadores que, automaticamente, aumentam o valor do operando. Exemplos:

PÓS-INCREMENTO	

`A = x++`	Substitui, nesta ordem, as expressões:
	`A = x;`
	`x = x + 1;`
	Para **`x=5`**, teremos primeiro **`A=5`** e, em seguida, o **`x`** passará a ter o valor **`6`**.

PRÉ-INCREMENTO	

`A = ++x`	Substitui, nesta ordem, as expressões:
	`x = x + 1;`
	`A = x;`
	Para `x=5`, teremos primeiro `x` passando a ter o valor `6` e, em seguida `A=6`.

4.3 - Operadores Decrementais

São operadores que, automaticamente, subtraem o valor do operando. Exemplos:

PÓS-DECREMENTO	

`A = x--`	Substitui, nesta ordem, as expressões:
	`A = x;`
	`x = x - 1;`
	Para `x=5`, teremos primeiro `A=5` e, em seguida, o `x` passará a ter o valor `4`.

PRÉ-DECREMENTO	

`A = --x`	Substitui, nesta ordem, as expressões:
	`x = x - 1;`
	`A = x;`
	Para `x=5`, teremos primeiro `x` passando a ter o valor `4` e, em seguida `A=4`.

4.4 - Operadores de Atribuição

O operador de atribuição é o igual (=), ou seja, em JavaScript significa atribuir e não igualar ou comparar:

OPERADORES DE ATRIBUIÇÃO	
`x = y` (Atribuição)	atribui o valor `y` à variável chamada `x`.
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x %= y` (Módulo)	`x = x % y` (resto da divisão de `x` por `y`) .

4.5 - Operadores de Comparação

Comparam os operandos, que podem ser valores **numéricos** ou ***strings***, e retornam um valor lógico `true` (verdadeiro) ou `false` (falso). São os operadores utilizados nas estruturas condicionais e laços lógicos:

OPERADORES DE COMPARAÇÃO	
`x == y` (Comparação de igualdades)	Retorna `true` se os operandos forem iguais.
`x === y` (Comparação de igualdades e tipos)	Retorna `true` se os operandos forem iguais e do mesmo tipo.
`x != y` (Comparação de desigualdades)	Retorna `true` se os operandos forem diferentes.
`x !== y` (Comparação de desigualdades e tipos)	Retorna `true` se os operandos forem diferentes no conteúdo e no tipo.
`x > y`	Retorna `true` se `x` for maior que `y`.
`x >= y`	Retorna `true` se `x` for maior ou igual a `y`.
`x < y`	Retorna `true` se `x` for menor que `y`.
`x <= y`	Retorna `true` se `x` for menor ou igual a `y`.

4.6 - Operadores Lógicos

Exigem valores **booleanos** como operandos, e retornam um valor lógico:

OPERADORES LÓGICOS	
`x && y` (E)	retorna `true` se as 2 expressões forem verdadeiras.
`x || y` (Ou)	retorna `false` se as 2 expressões forem falsas.
`!x` (Não)	retorna `false` se a expressão for verdadeira e retorna `true` se a expressão for falsa.

4.7 - Operador de String

É usado para concatenar (agrupar) dois ***strings***, através do sinal de "+", por exemplo:

```
<SCRIPT>
  A = "José";
  B = "Silva";
  C = A+ " da " +B;
  document.write("O nome é " +C);
</SCRIPT>
```

```
O nome é José da Silva
```

Podemos realizar a mesma operação da seguinte forma:

```
<SCRIPT>
  A = "José";
  B = "Silva";
  document.write("O nome é " A+ " da " +B);
</SCRIPT>
```

```
O nome é José da Silva
```

PARTE 03 - Escrevendo Scripts

5 - Visão Geral

Tendo em vista que o JavaScript é apenas texto sem formatação, você pode digitar os *scripts* em qualquer editor de texto desde que seja salvo no formato padrão texto, e com a extensão **`.htm`** ou **`.html`**. Veremos mais adiante que é possível produzir arquivos que contém apenas códigos JavaScript, neste caso, o documento deve ser salvo com a extensão **`.js`**.

As linhas de comentários devem iniciar com duas barras (**`//`**). Para comentários em mais de uma linha, deve iniciar com barra-asterísco (**`/*`**) e encerrar com asterísco-barra (**`*/`**).

5.1 - A *Tag* `<SCRIPT>...</SCRIPT>`

A *tag* de abertura de *script* informa ao navegador que em seguida serão executados *scripts*. O parâmetro **`language`** determina qual a linguagem de *script* será usada; por *default*, será JavaScript:

```
<!DOCTYPE html>
<HTML>
<HEAD>
   <TITLE>JavaScript - Exemplo 01</TITLE>
   <SCRIPT language = "JavaScript">
         document.title = "Exemplo 1";
   </SCRIPT>
</HEAD>
<BODY>
   <SCRIPT language = "JavaScript">
         document.bgColor = "yellow";
         document.fgColor = "darkblue";
         document.write("Funciona!!!");
   </SCRIPT>
</BODY>
</HTML>
```

Funciona!!!

No exemplo acima, será exibido na tela apenas a frase **`Funciona !!!`**, com **fundo** na cor amarela e **fonte** na cor azul-escura:

5.2 - A *Tag* `<NOSCRIPT>...</NOSCRIPT>`

Informa ao usuário da página que é preciso habilitar o JavaScript em seu navegador para, em seguida, executar os *scripts*.

```
<!DOCTYPE html>
<HTML>
<HEAD>
   <TITLE>JavaScript - Exemplo 01 </TITLE>
   <SCRIPT language = "JavaScript">
         document.title = "Exemplo 1";
   </SCRIPT>
</HEAD>
<BODY>
   <SCRIPT language = "JavaScript">
         document.bgColor = "yellow";
         document.fgColor = "darkblue";
         document.write("Funciona!!!");
   </SCRIPT>
<NOSCRIPT>
   <H2>Por gentileza, habilite o JavaScript!</H2>
</NOSCRIPT>

</BODY>
</HTML>
```

Por gentileza, habilite o JavaScript!

6 - Caixas de Diálogo

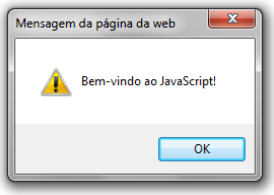
As caixas de diálogo são muito utilizadas em JavaScript, pois permitem exibir mensagens e solicitar decisões aos usuários.

6.1 - Caixas de Alerta

Esta caixa é gerada através do método **`alert()`**, que pertence ao objeto **`window`**. Na tela aparece uma pequena janela contendo a mensagem escrita pelo desenvolvedor e um botão com o rótulo **`[OK]`**.

```
<!DOCTYPE html>
<HTML>
<HEAD>
   <TITLE>Exemplo 02 - Alerta</TITLE>
   <SCRIPT language = "JavaScript">
        window.alert("Bem-vindo ao JavaScript!");
   </SCRIPT>
</HEAD>

<BODY>
   <H1>Exemplo de Caixa de Alerta</H1>
</BODY>
</HTML>
```

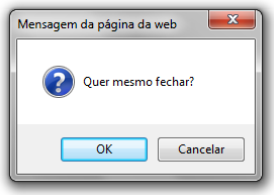


6.2 - Caixas de Confirmação

Esta caixa é gerada através do método **`alert()`** que pertence ao objeto **`window`**. Na tela aparece uma pequena janela contendo mensagem escrita pelo desenvolvedor e dois botões, um com o rótulo **`[OK]`** e o outro com o rótulo **`[CANCELAR]`**. Retorna **`true`** se o usuário clicar em **`[OK]`** e **`false`** se o usuário clicar em **`[CANCELAR]`**. Para tanto, o método **`alert()`** deve ser escrito dentro de uma estrutura condicional, como **`if()`** por exemplo:

```
<!DOCTYPE html>
<HTML>
<HEAD>
   <TITLE>Exemplo 03 - Confirmação</TITLE>
</HEAD>

<BODY>
   <H1>Exemplo de Caixa de Confirmação </H1><P>
   <SCRIPT language = "JavaScript">
      if (window.confirm("Quer mesmo fechar?"))
         {
         window.close();
         }
      else
         {
         document.write("Você não fechou!");
         }
   </SCRIPT>
</BODY>
</HTML>
```



PARTE 04 - Condicionais e Laços Lógicos

7 - Estruturas Condicionais

As estruturas condicionais se dividem entre as funções **`if()`** e **`switch()`**, como na maioria das linguagens de programação. São usadas para executar diferentes ações com base em condições diferentes, provocando desvios na execução do código, com base na verificação de expressões como **`true`** ou **`false`**.

7.1 - Condicional com `if()`

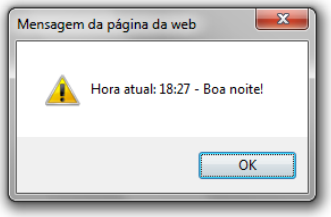
Usa-se **`if(condição)`** para especificar um bloco de instruções a ser executado, se uma condição específica for **verdadeira**. Usa-se **`else`** para especificar um bloco de código a ser executado se a mesma condição for **falsa**. As instruções devem ser escritas dentro de chaves **`{}`**, mas não é obrigatória a finalização de cada instrução com ponto-e-vírgula.

```
if (condição) {
   // bloco de instruções da parte verdadeira
   }
else {
   // bloco de instruções da parte falsa
   }
```

Pode-se também montar uma única estrutura **`if()`** com várias condições. Neste caso, para cada nova condição se utiliza **`else if()`**, mantendo o **`else`** final para o caso de nenhuma condição descrita ser avaliada como **`true`**. Não tem uma quantidade limite para a utilização de **`else if()`** na mesma estrutura condicional:

```
if (condição1) {
   // instruções para condição1 verdadeira
   }
else if (condição2) {
   // instruções para condição2 verdadeira
   }
else {
   // instruções para nenhuma condição verdadeira
   }
```

```
<SCRIPT language = "JavaScript">
  var hora  = new Date().getHours();
  var min   = new Date().getMinutes();
  var resp  = "Hora atual:" +hora+ ":" +min+ " - ";
    if (hora < 12) {
        resp += "Bom dia!";
    } else if (hora < 18) {
        resp += "Boa tarde!";
    } else {
        resp += "Boa noite!";
    }
  window.alert(resp);
</SCRIPT>
```

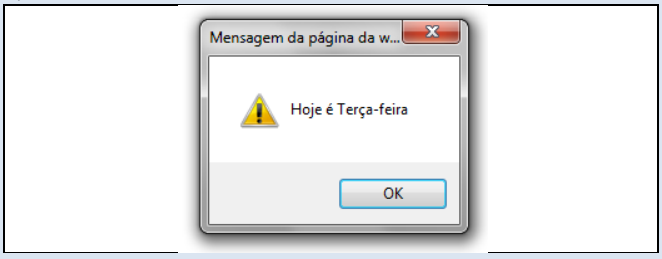


7.2 - Condicional com `switch()`

Usa-se **`switch(expressão)`** para especificar diferentes blocos de instruções a serem executados, a partir da análise de uma única variável que é analisada no início da estrutura. A expressão é comparada com os valores apresentados em cada **`case`**. Ao final de cada bloco de instruções deve-se declarar o comando **`break`** para que a execução das instruções seja desviada para o final da estrutura **`switch()`**. Usa-se **`default`** para especificar um bloco de código a ser executado se todos os casos forem **falsos**. Todos os casos devem ser escritos dentro de um único par de chaves **`{}`**, mas não é obrigatória a finalização de cada instrução com ponto-e-vírgula.

```
switch(expressão) {
   case n1:
      // caso variável = n1;
      break;
   case n2:
     // caso variável = n2;
      break;
   default:
      // nenhum caso verdadeiro;
}
```

```
<SCRIPT language = "JavaScript">
  var resp = "Hoje é ";
  switch (new Date().getDay())
  {
     case 0:
         resp += "Domingo";
         break;
     case 1:
         resp += "Segunda-feira";
         break;
     case 2:
         resp += "Terça-feira";
         break;
     case 3:
         resp += "Quarta-feira";
         break;
     case 4:
         resp += "Quinta-feira";
         break;
     case 5:
         resp += "Sexta-feira";
         break;
     case 6:
         resp += "Sábado";
  }
window.alert(resp);
</SCRIPT>
```



Editar o EXERCÍCIO 01 do Caderno de Exercícios

Editar o EXERCÍCIO 02 do Caderno de Exercícios

Editar o EXERCÍCIO 03 do Caderno de Exercícios

7.3 - Condicional Ternária

Pode-se também utilizar a forma alternativa de escrever uma estrutura condicional, utilizando três operadores na mesma instrução. A condicional ternária inicia com uma **`(condição)`** não necessariamente entre parênteses, seguida do ponto de interrogação. Depois especificam-se a partes executadas se a condição for verdadeira ou falsa, separadas por dois pontos.

```
(condição) ? parteVerdadeira : parteFalsa
```

É possível declarar mais de uma instrução nas partes verdadeiras e falsas, desde que estejam separadas por vírgulas mas dentro de um par de parênteses **`()`**.

```
(condição) ? (
        // instrução1 para verdadeira,
        // instrução2 para verdadeira
):(
        // instrução1 para falsa,
        // instrução2 para falsa )
```

```
<SCRIPT language = "JavaScript">
  var idade = 23;
  idade > 18 ?  (
     document.fgColor = "darkgreen",
     document.write("OK, você pode entrar!")
 ):(
     document.fgColor = "red",
     document.write("Desculpe, você é muito jovem!")
     )
</SCRIPT>
```

8 - Laços Lógicos (*Loops*)

Os laços lógicos são estruturas que executam um bloco de instruções por uma determinada quantidade de vezes, em repetição, são os chamados *Loops*. Para tanto, utiliza-se o comando **`for()`**, **`for()/in`**, **`while()`** e **`do/while()`**.

8.1 - *Loop* com `for()`

Esse tipo de *loop* utiliza um contador (uma variável com um valor incial) e termina quando um teste condicional for satisfeito. O comando **`for()`** inicia a estrutura, tendo dentro dos parênteses a definição da **variável contadora**, da **condição de término** e do **incremento** a ser aplicado à cada repetição, todos separados por ponto-e-vírgula. O bloco de instruções a ser executado enquanto a condição é verdadeira deve ser escrito dentro de chaves **`{}`**. Veja a sintaxe:

```
for (contador; condição; incremento)
        {
        // enquanto a condição é verdadeira
        }
// após a condição tornar-se falsa
```

```
<SCRIPT language = "JavaScript">
   for (i=1; i<6; i++)
         {
         document.write(i + "<br>");
         }
   document.write("Acabou!");
</SCRIPT>
```

```
1
2
3
4
5
Acabou!
```

8.1 - *Loop* com `while()`

Esse tipo de *loop* utiliza uma expressão como condição de execução do bloco de instruções. O bloco se repetirá enquanto esta condição é verdadeira. Uma expressão é declarada dentro dos parênteses de **`while(expressão)`** O bloco de instruções para repetição deve ser escrito dentro de uma par de chaves **`{}`**. Caso seja necessária uma **variável contadora**, esta deve ser declarada antes de iniciar a estrutura **`while()`**. Caso seja necessário um **incremento** a ser aplicado à cada repetição, este deve ser escrito como uma das instruções a ser executada dentro das chaves **`{}`**. Veja a sintaxe:

```
contador;
for (condição)
        {
        // enquanto a condição é verdadeira
        incremento;
        }
// após a condição tornar-se falsa
```

```
<SCRIPT language = "JavaScript">
   var i=1;
         while(i<6)
         {
         document.write(i + "<br>");
         i++;
         }
   document.write("Acabou!");
</SCRIPT>
```

```
1
2
3
4
5
Acabou!
```

Editar o EXERCÍCIO 04 do Caderno de Exercícios

Editar o EXERCÍCIO 05 do Caderno de Exercícios

Editar o EXERCÍCIO 06 do Caderno de Exercícios

Editar o EXERCÍCIO 07 do Caderno de Exercícios