

Noções de JavaScript

Programação Web

Roteiro da Aula

- O que é JavaScript
- Formas de usar JavaScript
- Sintaxe da linguagem
- Relacionando elementos HTML no código JavaScript
- Funções nativas
- Exemplos práticos

O que é JavaScript

- **JAVASCRIPT É UMA LINGUAGEM** de programação interpretada criada em 1995, como uma extensão do HTML para o browser Navigator 2.0. Hoje existem implementações JavaScript nos browsers dos principais fabricantes.
- **Programas em JavaScript** são interpretados linha-por-linha enquanto o browser carrega a página ou executa uma rotina.

O que é JavaScript

- **JavaScript é baseada em objetos.** Trata suas estruturas básicas, propriedades do browser e os elementos de uma página HTML como objetos (entidades com propriedades e comportamentos) e permite que sejam manipulados através de eventos do usuário programáveis, operadores e expressões

O que se pode fazer com JavaScript?

- Com JavaScript pode-se fazer diversas coisas que não é possível com HTML:
 - Realizar operações matemáticas e computação.
 - Abrir janelas do browser, trocar informações entre janelas, manipular propriedades do browser como o histórico, barra de estado, plug-ins e applets.
 - Interagir com o conteúdo do documento, alterando propriedades da página, dos elementos HTML e tratando toda a página como uma estrutura de objetos.
 - Interagir com o usuário através do tratamento de eventos

Formas de usar JavaScript

- Para editar código JavaScript, é necessário apenas um simples editor de texto, como o Bloco de Notas do Windows.
- Pode-se também usar um editor HTML. Alguns editores colocam cores ou dão destaque ao código JavaScript. Outros até permitem a geração de código ou a verificação de sintaxe.

Formas de usar JavaScript

- Há três maneiras de incluir JavaScript em uma página Web:
 - Dentro de blocos HTML `<script> ... </script>` em várias partes da página.
 - Em um arquivo externo, importado pela página: para definir funções que serão usadas por várias páginas de um site.
 - Dentro de descritores HTML sensíveis a eventos: para tratar eventos do usuário em links, botões e componentes de entrada de dados, durante a exibição da página.

Formas de usar JavaScript

- A forma mais prática de usar JavaScript é embutindo o código na página dentro de um bloco delimitado pelos descritores HTML **<script>** e **</script>**.
- Pode haver vários blocos **<script>** em qualquer lugar da página.

```
<script>  
... instruções JavaScript ...  
</script>
```


Formas de usar JavaScript

- O descritor **<script>** possui um atributo **language** que informa o tipo e versão da
- linguagem utilizada. O atributo **language** é necessário para incluir blocos em outras linguagens como VBScript. **É opcional para JavaScript:**

```
<script language="vbscript"> ...código em vbscript...
```

```
</script>
```

```
<script language="javascript"> ...código javascript...
```

```
</script>
```

```
<script> ... código javascript ... </script>
```

Formas de usar JavaScript

- Muitas vezes é necessário realizar um mesmo tipo de tarefa mais de uma vez. Para esse tipo de problema JavaScript permite que o programador crie funções que podem ser chamadas de outras partes da página várias vezes.
- Se várias páginas usam as mesmas funções JavaScript definidas pelo autor da página, uma boa opção é colocá-las em um arquivo externo e importá-lo nas páginas que precisarem delas.

Formas de usar JavaScript

- Quando utilizamos o javascript na web temos que desenvolver nossos códigos dentro de funções
- Funções encapsulam o código e podem ser invocados/chamados a qualquer momento dentro do seu código
- Alguns benefícios do uso: clareza do código, reutilização e independência

Formas de usar JavaScript

- Este arquivo deve ter a extensão “.js” e conter apenas código JavaScript (não deve ter descritores HTML, como **<script>**). Por exemplo, suponha que o arquivo meu_codigo.js possua o seguinte código JavaScript:

```
function soma(a, b)
{
    return a + b;
}
```

Formas de usar JavaScript

- Para carregar esta função e permitir que seja usada em outra página, usa-se o atributo **src** do descritor **<script>**:

```
<script LANGUAGE=JavaScript SRC=« meu_codigo.js" >  
resultado = soma(5, 6);  
document.write('<p>A soma de 5 e 6 é ` + resultado</p>');  
</script>
```

Formas de usar JavaScript

- A linguagem JavaScript permitem a captura de eventos disparados pelo usuário, como o arrasto de um mouse, o clique de um botão, etc.
- Quando ocorre um evento, um procedimento de manuseio do evento é chamado. O que cada procedimento irá fazer pode ser determinado pelo programador.

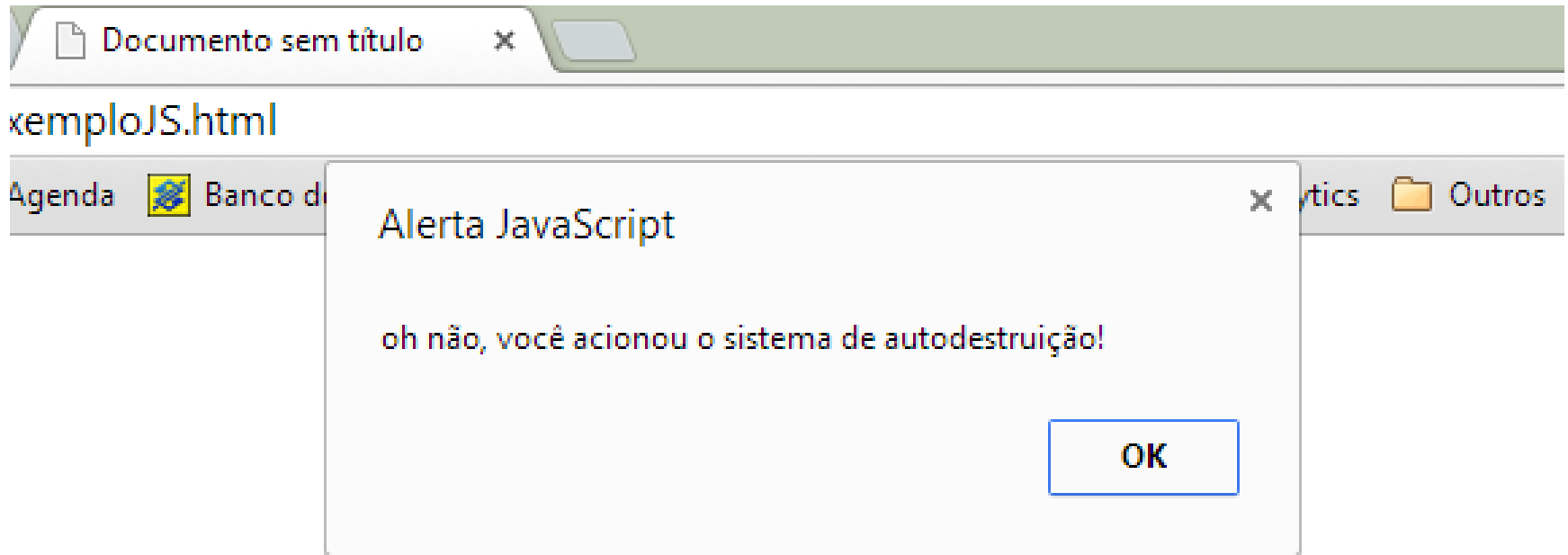
Formas de usar JavaScript

- Os atributos de eventos se aplicam a elementos HTML específicos e sempre começam com o prefixo “on”. Os valores recebidos por esses atributos é código JavaScript. Por exemplo:

```
<form>  
<input type="button"  
onclick="alert('oh não, você acionou o sistema de  
autodestruição!')"  
value="não aperte este botão">  
</form>
```

Formas de usar JavaScript

- Execução do script do slide anterior:



Formas de usar JavaScript

- Tudo o que aparece entre as aspas duplas do atributo **onclick** é JavaScript. **onclick** é um atributo HTML, criado como extensão para dar suporte ao evento de clicar o botão.
- O código JavaScript que está em negrito será interpretado quando o usuário apertar o botão com o mouse (**onclick**).
- A instrução **alert()** cria uma janela de alerta (acima) com a mensagem passada como parâmetro (entre parênteses e aspas no código em negrito).

Formas de usar JavaScript

- Os procedimentos de manuseio de eventos introduzidos por JavaScript são:

Atributo HTML	Quando o procedimento é executado	Descritores HTML onde é suportado
onclick	Quando um objeto é clicado pelo mouse	<a>, <input>
onselect	Quando um objeto é selecionado	<input type=text>, <textarea>
onchange	Quando uma seleção ou campo de texto tem seu conteúdo modificado	<input type=text>, <textarea>, <select>
onfocus	Quando um componente de formulário ou janela se torna ativa	<textarea>, <body>, <form>, <input>, <select>, <option>
onblur	Quando um componente de formulário ou janela se torna inativa	<textarea>, <body>, <form>, <input>, <select>, <option>

Formas de usar JavaScript

- Procedimentos de manuseio de eventos:

onmouseover	Quando o mouse está sobre um link	<a>, <area>
onmouseout	Quando o mouse deixa um link	<a>, <area>
onsubmit	Antes de enviar um formulário	<input type=submit>
onreset	Antes de limpar um formulário	<form>
onload	Após carregar uma página, janela ou frame	<body>
onunload	Ao deixar uma página, janela ou frame	<body>
onerror	Quando um erro ocorre durante a carga de uma imagem ou página	, <body>
onabort	Quando a carga de uma imagem é abortada	

Formas de usar JavaScript

- Como procedimentos de eventos são atributos do HTML (e não do JavaScript), tanto faz escrevê-los em letras maiúsculas ou minúsculas.
- Usar onclick, ONCLICK ou OnClick não faz diferença.
- Já o texto dentro das aspas é JavaScript, que é uma linguagem que diferencia letras maiúsculas de minúsculas, portanto alert não é a mesma coisa que ALERT.

Resumo de funções da aula

Função	Objetivo
<code>document.write('mensagem')</code>	Escreve uma mensagem na página web aberta.
<code>alert('mensagem')</code>	Exibe uma janela de mensagem de alerta na página

Sintaxe e Estrutura

Sintaxe e estrutura

- Como a maior parte das linguagens de programação, o código JavaScript é expresso em formato texto.
- O texto do código pode representar instruções, grupos de instruções, organizadas em blocos e comentários.
- Dentro das instruções, pode-se manipular valores de diversos tipos, armazená-los em variáveis e realizar diversas operações com eles.

Sintaxe e estrutura

- Instruções compostas (seqüências de instruções que devem ser tratadas como um grupo) são agrupadas em blocos delimitados por chaves ({ e }), como mostrado abaixo:

```
function xis() {  
  var x = 0;  
  while (x < 10) {  
    x = x + 1;  
  }  
}
```


Sintaxe e estrutura

- Blocos são usados em JavaScript na definição de funções e estruturas de controle de fluxo.
- Blocos são tratados como uma única instrução e podem ser definidos dentro de outros blocos.
- No exemplo anterior, o bloco da função **xis()** contém duas instruções. A segunda é um bloco (**while**) que contém uma instrução. As outras instruções não pertencem a bloco algum.

Sintaxe e estrutura

- As chaves que definem um bloco são caracteres separadores. Podem ser colocadas em qualquer lugar após a declaração da estrutura que representam.
- Não precisam estar na mesma linha. Pode haver qualquer número de caracteres terminadores de linha antes ou depois:

```
function media(a, b)
{
return (a + b)/2;
}
```

Sintaxe e estrutura

- Os formatos maiúsculo e minúsculo de um caractere são considerados caracteres distintos em JavaScript.
- Por exemplo `function`, `Function` e `FUNCTION` são três nomes distintos e tratados diferentemente em JavaScript.

Sintaxe e estrutura

- Há duas formas de incluir comentários em JavaScript.
- Qualquer texto que aparece depois de duas barras (//) é ignorado pelo interpretador até o final da linha.
- Quando o interpretador encontra os caracteres `/*`, ignora tudo o que aparecer pela frente, inclusive caracteres de nova-linha, até encontrar a sequência `*/`.

Sintaxe e estrutura

■ Comentários:

```
/* Esta função retorna a  
* média dos argumentos passados  
*/  
function media(a, b)  
{  
  return (a + b)/2; // a e b devem ser números  
}
```

Variáveis

- Variáveis são usadas para armazenar valores temporários na maior parte das instruções em JavaScript. Para definir uma variável, basta escolher um nome que não seja uma palavra reservada e lhe atribuir um valor:

```
preco = 12.6;  
produto = "Livro";
```

- Uma variável também pode ser declarada sem que receba um valor. Para isto é necessário usar a palavra-chave var:

```
var preco;
```

Variáveis

- A linguagem não é rigorosa em relação a tipos de dados e portanto não é preciso declarar os tipos das variáveis antes de usá-las, como ocorre em outras linguagens.
- O tipo de dados é alocado no momento da inicialização, ou seja, se na definição de uma variável ela receber uma string, JavaScript a tratará como string até que ela receba um novo tipo através de outra atribuição.

Variáveis

- O escopo ou alcance de uma variável depende do contexto onde é definida ou declarada.
- Uma variável declarada ou definida pela primeira vez dentro de um bloco tem escopo local ao bloco e não existe fora dele.
- Variáveis declaradas ou definidas fora de qualquer bloco são globais e são visíveis em todo o programa ou página HTML.

Variáveis

- Exemplo variáveis globais:

```
<script>
global = 3; // escopo: toda a pagina
function func() {
  local = 5; // escopo: somente o bloco atual
  global = 10;
}
// local nao existe aqui.
// global tem valor 10! (pode ser lida em qualquer lugar
da pagina)
</script>
```

Variáveis

- O uso de `var` é opcional na definição de variáveis globais. Variáveis locais devem ser definidas com `var` para garantir que são locais mesmo havendo uma variável global com o mesmo nome, por exemplo:

```
g = 3; // variável global
function func() {
var g = 10; // esta variável g é local!
}
// g (global) tem o valor 3!
```

Variáveis

- Identificadores JavaScript são os nomes que o programador pode escolher para variáveis e funções definidas por ele. Esses nomes podem ter qualquer tamanho e só podem conter caracteres que sejam:
 - números (0-9)
 - letras (A-Z e a-z)
 - caractere de sublinhado (_)
- Além disso, embora identificadores JavaScript possam conter números, não podem *começar com número*.
- Existem ainda algumas palavras que são reservadas da linguagem, e estas não podem ser usadas como identificadores de variáveis

Operadores e Expressões

- JavaScript possui várias classes de operadores.
- Operações de atribuição, aritméticas, booleanas, comparativas e binárias em JavaScript são realizadas da mesma forma que em outras linguagens estruturadas como C++ ou em Java.

Operadores Aritméticos

Operador	Função
<code>++n,n++</code>	Incremento
<code>--n,n--</code>	Decremento
<code>*</code>	Multiplicação
<code>/</code>	Divisão
<code>%</code>	Resto
<code>+</code>	Adição e concatenação
<code>-</code>	Subtração

Operadores lógicos e relacionais

Operador	Função
!=	Diferente de
==	Igual a
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
	Ou (or)
&&	E (and)
!	Negação (not)

Estruturas de controle de fluxo

- As estruturas de controle de fluxo são praticamente as mesmas utilizadas em outras linguagens estruturadas populares.
- **if...else**: A estrutura if... else é utilizada para realizar controle de fluxo baseado em expressões condicionais:

```
if (condição) {  
    // instruções caso condição == true  
} else if (condição 2) {  
    // instruções caso condição 2 == true  
} else {  
    // instruções caso ambas as condições sejam false  
}
```

Estruturas de controle de fluxo

■ Exemplo:

```
if (ano < 0) {  
  alert("Digite um ano D.C.");  
} else if ( ((ano % 4 == 0) && (ano % 100 != 0)) || (ano  
% 400 == 0)) {  
  alert(ano + " é bissexto!");  
} else {  
  alert(ano + " não é bissexto!");  
}
```


Estruturas de controle de fluxo

- ***for***: As estruturas **for** e **while** são usadas para repetições baseadas em condições. O **bloco for** contém de três parâmetros : uma inicialização, uma condição e um incremento. A sintaxe é a seguinte:

```
for(inicialização; condição; incremento) {  
  // instruções a serem realizadas enquanto condição for  
  true  
}
```

Estruturas de controle de fluxo

- Por exemplo:

```
for (i = 0; i < 10; i = i + 1) {  
  document.write("<p>Linha " + i + "</p>");  
}
```

- A primeira coisa realizada no bloco for é a inicialização. É feita uma vez apenas.
- A condição é testada cada vez que o loop é reiniciado. O incremento é realizado no final de cada loop.

Estruturas de controle de fluxo

- ***while***: O mesmo que foi realizado com for pode ser realizado com uma estrutura **while**, da forma:

```
inicialização;  
while(condição) {  
  // instruções a serem realizadas enquanto condição for  
  true  
  incremento;  
}
```

Estruturas de controle de fluxo

- Veja como fica o mesmo exemplo acima usando **while**:

```
i = 0
while (i < 10) {
  document.write("<p>Linha " + i + "</p>");
  i++;
}
```

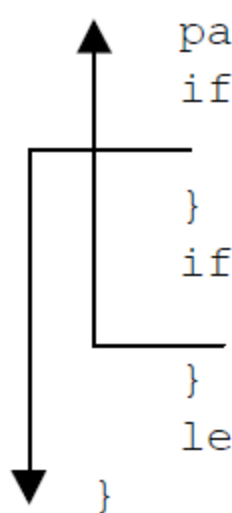
Estruturas de controle de fluxo

- **break e continue:** Para sair a força de loops em cascata existem ainda as instruções **break e continue**.
- **break** : sai da estrutura de loops e prossegue com a instrução seguinte.
- **continue** : deixa a execução atual do loop e reinicia com a passagem seguinte.

Estruturas de controle de fluxo

- Exemplo break e continue:

```
function leiaRevista() {  
  while (!terminado) {  
    passePagina();  
    if (alguemChamou) {  
      break;           // caia fora deste loop (pare de ler)  
    }  
    if (paginaDePropaganda) {  
      continue; // pule esta iteração (pule a pagina e nao leia)  
    }  
    leia();  
  }  
}
```



Vetores/Arrays

- O tipo Array representa coleções de qualquer tipo, na forma de vetores ordenados e indexados.
- Para criar um novo vetor em JavaScript, é preciso usar o operador new e o construtor Array():

```
direcao = new Array(4);
```

Vetores/Arrays

- Vetores começam em 0(zero) e terminam em **length-1**. **length** é a única propriedade do tipo Array.
- Contém um número com o comprimento do vetor. Os elementos do vetor são acessíveis através de índices passados entre colchetes ([e]).
- Para acessar qualquer um dos elementos do vetor *direcao*, por exemplo, usa-se o nome da variável seguida do índice do elemento entre colchetes:

```
x = direcao[2]; // copia o conteúdo do terceiro elemento  
de direcao em  
x
```


Manipulando objetos do HTML

Manipulando objetos do HTML

- Todos os objetos criados em HTML estão automaticamente disponíveis em JavaScript, mesmo que um nome não seja atribuído a eles.
- Por exemplo, se há três blocos `<form> . . . </form>` em uma página, há três objetos do tipo Form no JavaScript

Manipulando objetos do HTML

- Se eles não tem nome, pode-se ter acesso a eles através da propriedade **forms** definida em **document**.
- Essa propriedade armazena os objetos Form em uma coleção ordenada de propriedades (vetor). Todos os índices usados nos vetores em JavaScript iniciam a contagem em 0, portanto, `document.forms[0]`, refere-se ao primeiro formulário de uma página.
- Cada formulário pode então ser recuperado através de seu índice:

```
frm1 = document.forms[0] ;  
frm2 = document.forms[1] ;
```

Manipulando objetos do HTML

- Se houver, por exemplo, dentro de um bloco `<FORM>...</FORM>` 5 componentes, entre botões, campos de texto e caixas de seleção, existirão 5 objetos em JavaScript dos tipos Text, Button e Select.
- Independente do tipo de componente de formulário, eles podem ser acessados na ordem em que aparecem no código, através da propriedade `elements`, de Form:

```
texto = document.forms[0].elements[1];
```

Manipulando objetos do HTML

- Os vetores são necessários apenas quando um objeto não tem nome. Se tiver um nome (definido no código HTML, através do atributo NAME do descritor correspondente), o ideal é usá-lo já que independe da ordem dos componentes, e pode fornecer mais informações como por exemplo, o tipo do objeto

Manipulação de objetos do HTML

- Exemplo de manipulação de objetos do HTML através do atributo **name**:

```
<form name="f1">  
<input type=button name="botao1" value="Botão 1">  
<input type=text name="campoTexto" value="Texto Muito Velho">  
</form>
```

- Agora é possível ter acesso ao campo de textos em JavaScript usando nomes, em vez de índices de vetores:

```
texto = document.f1.campoTexto;  
textoVelho = texto.value; // lendo a propriedade value...  
texto.value = "Novo Texto";
```

Manipulação de objetos do HTML

- O código do slide anterior também poderia ter sido escrito da forma, com os mesmos resultados:

```
textoVelho = document.f1.campoTexto.value;  
document.f1.campoTexto.value = "Novo Texto";
```

Manipulação de objetos do HTML

- Exemplo em sala de aula: Somador

Somador JavaScript

<input type="text"/>	+	<input type="text"/>	=	<input type="text"/>
----------------------	---	----------------------	---	----------------------

Somador JavaScript

- Código JavaScript:
- OBS: A função **parseFloat** converte uma string para um valor numérico com casa decimal.

```
<script language=JavaScript>
function soma() {
a = document.f1.val1.value;
b = document.f1.val2.value;
document.f1.val3.value = parseFloat(a) + parseFloat(b);
}
</script>
```

Somador JavaScript

- Código HTML:

```
<body>
<h1>Somador JavaScript</h1>
<form name="f1">
<input type=text name="val1" size="5"> +
<input type=text name="val2" size="5">
<input type=button value="somar" onclick="soma()">
<input type=text name="val3" size="5">
</form>
</body>
```

Funções nativas

Funções Nativas

- JavaScript possui várias funções e objetos nativos, que são procedimentos que permitem realizar tarefas úteis no dia-dia, como conversão de tipos, cálculos com datas, funções matemáticas, entre outras.
- Todas recebem parâmetros com os dados sobre os quais devem operar. Podem ser chamadas de qualquer lugar. Por exemplo:

```
ano = parseInt("2010");
```

parseInt(string)

- Converte uma String para a sua representação numérica. Ignora qualquer coisa depois do ponto decimal ou depois de um caractere que não é número.
- Se primeiro caractere não for número, retorna NaN (Not a Number).

```
ano = parseInt("2010");
```

parseFloat(string)

- Converte uma *String* para a sua *representação numérica*, levando em *consideração o ponto decimal*.
- Ignora qualquer coisa depois do segundo ponto decimal ou depois de um caractere que não é número.
- Se primeiro caractere não for número ou ponto decimal, retorna NaN (Not a Number)

```
ano = parseFloat("2.98") ;
```

isNaN(valor)

- Retorna true se o valor passado não é um número.

```
if (!isNaN(valor))  
{  
  document.write('O valor informado não é um campo  
numérico');  
}
```

Objeto string

- O tipo String existe para dar suporte e permitir a invocação de métodos sobre cadeias de caracteres, representadas pelo tipo primitivo string.
- Pode-se criar um novo objeto String fazendo:

```
s = new String("string");
```

ou simplesmente:

```
s = "string";
```


Objeto string

- Objetos String possuem apenas uma propriedade: **length**, que pode ser obtida a partir de qualquer objeto string e contém o comprimento da cadeia de caracteres:

```
tamanho = s.length;
```

Objeto string

- Os dois métodos a seguir realizam transformações no formato dos caracteres. São extremamente úteis em comparações e rotinas de validação. Retornam *String*.

Método Invocado	Retorno	Exemplo
toLowerCase()	texto (converte para caixa-baixa)	valor = valor.toLowerCase();
toUpperCase()	TEXTTO (converte para caixa-alta)	valor = valor.toUpperCase ();

Objeto string

- Os métodos seguintes realizam operações baseados nos caracteres individuais de uma string. Não afetam os strings originais. As transformações são retornadas.

Método Invocado	Retorno	Exemplo
<code>charAt(n)</code>	Retorna o caractere na posição <i>n</i> .	<code>primeiraLetra = valor. charAt(0);</code>
<code>indexOf("substring")</code>	<i>Retorna um índice n referente à posição da primeira ocorrência de "substring" na string s.</i>	<code>indice = valor. indexOf("s");</code>
<code>lastIndexOf("substring")</code>	Retorna um índice <i>n</i> referente à posição da última ocorrência de "substring" na string <i>s</i> .	<code>indice = valor. lastIndexOf("s");</code>

Objeto string

Método Invocado	Retorno	Exemplo
<code>split("delimitador")</code>	Converte o string em um vetor de strings separando-os pelo "delimitador" especificado.	<pre>data = "Sexta-feira, 13 de Agosto de 1999"; sexta = data.split(",");</pre>
<code>substring(inicio, fim)</code>	Extraí uma substring de uma string s. <ul style="list-style-type: none">• inicio é um valor entre 0 e s.length-1.• fim é um valor entre 1 e s.length. O caractere na posição inicio é incluído na string e o caractere na posição fim não é incluído. A string resultante contém caracteres de inicio a fim -1.	<pre>pedaco= valor. substring(0,2);</pre>

Objeto Math

- O objeto Math contém um conjunto de funções matemáticas. Para ter acesso a suas funções e constantes, deve-se usar a sintaxe:

```
Math.função();
```

Objeto Math

- As funções do tipo Math estão listados na tabela a seguir:

Funções			
acos (x)	coosseno ⁻¹	abs (x)	absoluto
asin (x)	seno ⁻¹	max (a, b)	máximo
atan (x)	tangente ⁻¹	min (a, b)	mínimo
atan2 (x, y)	retorna o ângulo θ de um ponto (x,y)	pow (x, y)	x^y
		sin (x)	seno
ceil (x)	arredonda para cima (3.2 e 3.8 \rightarrow 4)	round (x)	arredonda (3.49 \rightarrow 3 e 3.5 \rightarrow 4)
cos (x)	coosseno	tan (x)	tangente
exp (x)	e^x	sqrt (x)	raiz quadrada
floor (x)	arredonda para baixo (3.2 e 3.8 \rightarrow 3)	log (x)	logarítmo natural
random ()	retorna um número pseudo-aleatório entre 0 e 1.		

Objeto Date

- O tipo Date é um tipo de objeto usado para representar datas. Para criar data que represente a data e hora atuais, chame-o usando new, da forma:

```
aquiAgora = new Date();
```

Objeto Date

- Para utilizar as informações de um Date, invoca-se os seus métodos sobre o objeto criado. Métodos podem ser invocados a partir de um objeto Date como no exemplo a seguir:

```
dia = umDia.getDay();  
hora = umDia.getHours();  
ano = umDia.getYear();  
document.writeln("Horário de Greenwich: " +  
umDia.toGMTString());
```


Objeto Date

- A tabela a seguir relaciona os métodos dos objetos do tipo Date, os tipos de retorno (se houver) e suas ações.

Método	Ação
<code>getDate ()</code>	Retorna <i>Number</i> . Recupera o dia do mês (1 a 31)
<code>getDay ()</code>	<i>Number</i> . Recupera o dia da semana (0 a 6)
<code>getHours ()</code>	<i>Number</i> . Recupera a hora (0 a 23)
<code>getMinutes ()</code>	<i>Number</i> . Recupera o minuto (0 a 59)
<code>getMonth ()</code>	<i>Number</i> . Recupera o mês (0 a 11)
<code>getSeconds ()</code>	<i>Number</i> . Recupera o segundo (0 a 59)
<code>getTime ()</code>	<i>Number</i> . Recupera a representação em milissegundos desde 1-1-1970 0:0:0 GMT
<code>getTimezoneOffset ()</code>	<i>Number</i> . Recupera a diferença em minutos entre a data no fuso horário local e GMT (não afeta o objeto no qual atua)
<code>getFullYear ()</code>	<i>Number</i> . Recupera ano menos 1900 (1997 → 97)

Objeto Date

Método	Ação
<code>setDate (dia_do_mês)</code>	Acerta o dia do mês (1 a 31)
<code>setHours (hora)</code>	Acerta a hora (0 a 23)
<code>setMinutes (minuto)</code>	Acerta o minuto (0-59)
<code>setMonth (mês)</code>	Acerta o mês (0-11)
<code>setSeconds ()</code>	Acerta o segundo (0-59)
<code>setTime ()</code>	Acerta a hora em milissegundos desde 1-1-1970 0:0:0 GMT
<code>setYear ()</code>	Acerta o ano (ano – 1900)
<code>toGMTString ()</code>	<i>String</i> . Converte uma data em uma representação GMT
<code>toLocaleString ()</code>	<i>String</i> . Converte a data na representação local do sistema

Exemplos práticos

Exemplos práticos

- Exemplos práticos de Javascript:
 - Abrir uma janela popup
 - Janela de alerta após o carregamento /saída da página
 - Mensagem de Confirmação
 - Cor de fundo
 - Janela pop-up programada
 - Hora certa na barra de status
 - Input text com texto padrão e valor em branco quando usuário clicar
 - Verificando caracteres ao digitar em uma input text

Abrir uma janela popup

- A **JANELA DO BROWSER** é manipulável de várias formas através da linguagem JavaScript. Pode-se alterar dinamicamente várias de suas características como tamanho, aparência e posição. Para abrir uma janela usamos o método **window.open()**. Veja exemplo:

```
window.open('URL', 'titulo da  
página', 'top=valor, left=valor, width=valor, height=valor');
```

```
<body  
onload="window.open('teste.html', 'pagina', 'top=400, left=4  
00, width=50, height=50');" >
```

Janela de alerta após o carregamento /saída da página

- Podemos inserir uma mensagem de alerta ao carregar uma página ou quando o usuário sair dela. Usamos os métodos **onunload** ou **onload** da tag **body**.

```
<body  
onload="alert('Bem vindo');"  
onunload="alert('Obrigado pela visita, volte sempre!');">
```

Mensagem de Confirmação

- Para exibir uma mensagem de confirmação usamos a função **confirm("Mensagem")**.
- Veja exemplo:

```
function abrirJanela() {  
    if (confirm("Tem certeza que deseja abrir esta  
página?"))  
    {  
        window.open("teste.html");  
    }  
}
```

```
<body  
onload="alert('Bem vindo');"  
onunload="alert('Obrigado pela visita, volte sempre!');">
```

Cor de fundo

- Podemos permitir que o usuário escolha a cor de fonte do site. Para isso iremos criar uma página com três inputs do tipo radio. Em cada input usaremos o método **onclick** com o código javascript abaixo:

```
document.bgColor='cor'
```

```
Azul <input TYPE="radio"
onClick="document.bgColor='blue' ">
Vermelho <input TYPE="radio"
onClick="document.bgColor='red' ">
Amarelo <input TYPE="radio"
onClick="document.bgColor='yellow' ">
```


Janela pop-up programada

- A função **setTimeout("instruções", atraso)** Executa uma ou mais instruções JavaScript após um período de atraso em milissegundos.
- A função retorna um número de identificação que pode ser passado como argumento do método **clearTimeout()** para executar a operação imediatamente, ignorando o tempo que falta.

Janela pop-up programada

- A função **clearTimeout(id)** cancela a temporização de uma operação **setTimeout()** cujo número de identificação foi passado como parâmetro, e faz com que as instruções do **setTimeout()** sejam interpretadas e executadas imediatamente.

Janela pop-up programada

- No exemplo a seguir é criada uma função que abre uma janela e fecha a mesma em 3 segundos.

```
function abrirJanela() {  
    janela = window.open("teste.html");  
    setTimeout('janela.close();', 3000);  
}
```

- Para executa-la, usaremos o método **onclick** da tag **body**.

```
<body onload="abrirJanela();">
```

Hora certa na barra de status

```
function iniciarrelogio(){
    setTimeout('mostrarhora();', 1000);}

function mostrarhora(){
    var hora = new Date();

    h = hora.getHours();
    m = hora.getMinutes();
    s = hora.getSeconds();
    window.status = "Olá bem vindo! A hora certa é:" +
h + ":" + m + ":" + s;

    setTimeout('mostrarhora();', 1000);}
```

```
<body onload="iniciarrelogio();" >
```

Input text com texto padrão e valor em branco quando usuário clicar

- Podemos aplicar um efeito simples mas bastante útil para passar informações ao usuário quando usarmos formulário.
- Neste exemplo usaremos um **input text**, com um texto padrão quando ele estiver inativo e em branco quando o usuário clicar no campo.

```
<label>Nome: </label>  
<input type="text" name="nome" onblur="this.value='Digite  
o nome completo';" onclick="this.value=' ' ;"/>
```

Verificando caracteres ao digitar em uma input text

- Podemos desenvolver código Javascript para realizar validações no momento que o usuário digitar alguma caractere em uma **input text**. Para isso, usaremos o método **onkeydown**.
- Para verificar qual o caractere digitado, teremos que recorrer a tabela ASC para verificar o valor da tecla e checar a condição.
- Cada tecla do teclado tem um código correspondente. Para saber o código de cada tecla acesse a [tabela aqui](#).

Verificando caracteres ao digitar em uma input text

- No exemplo abaixo, a validação é feita para permitir que apenas caracteres numéricos sejam aceitos na entrada de dados.

```
<label>Nome: </label>  
<input type="text" name="nome"  
onkeydown="if (event.keyCode< 48 || event.keyCode >  
57){alert('Você só pode digitar números neste campo.');
```

