Aula 14: Os Objetos `Array, Date` e `Math`

Nesta aula continuaremos a tratar dos objetos em JavaScript. Veremos agora detalhes, propriedades e métodos dos objetos embutidos `Array, Date` e `Math`.

Objetivos:
Aprender as características e métodos dos objetos:
- Array,
- Math e
- Date.

Pré-requisitos:
A aula 13 que introduziu os conceitos de objeto e a aula 12, principalmente os conceitos de funções e variáveis que foram explicados.

1. O objeto `Array`

O objeto embutido `Array` (como os demais objetos) é uma coleção de dados. Mas, enquanto os objetos identificam cada dado por um nome, no `Array` cada dado tem como referência um número ou índice. Diz-se que os *arrays* permitem criar variáveis indexadas capazes de armazenar um conjunto de valores.

Em computação, *array* é um arranjo ou estrutura ordenada contendo elementos acessíveis individualmente referenciados por números, usados para armazenar tabelas ou conjuntos de dados relacionados e freqüentemente do mesmo tipo.

Antes de utilizar um *array* é preciso criá-lo utilizando o operador `new`, indicando o seu tamanho (número de elementos que pode guardar). Para acessar cada elemento de um *array* utiliza-se um índice entre colchetes após o nome da variável. Em algumas linguagens de programação, o primeiro elemento tem índice 1, mas em JavaScript, como em C, C++ e Java, a primeira posição (como no caso das *strings* que vimos na seção passada) é sempre 0.

O exemplo a seguir mostra a criação de um `array` de 4 posições e a atribuição de valores a estas posições:

```
valores = new Array(4);
valores[0] = 390;
valores[1] = 40;
valores[2] = 100;
valores[3] = 5;
```

Na maioria das linguagens (Pascal, C, Java), os *arrays* têm

um tamanho fixo que é especificado na sua criação. Mas, em JavaScript, o tamanho do `Array` pode ser posteriormente modificado, isto é, pode ser alterado dinamicamente. O tamanho pode ser alterado a qualquer momento simplesmente adicionando mais um elemento.

Não é necessário usar apenas valores constantes na referência aos índices dos *arrays*. Uma expressão arbitrária qualquer que resulte valores não negativos pode ser usada, como nas linhas seguintes:

```
i=2;
valores[i+i] = 25;
valores[valores[3]] = 15;
```

Os *arrays* podem conter qualquer tipo de dado, incluindo um outro `Array`, um objeto ou função. Por exemplo: o código a seguir se refere à propriedade `width`, do segundo elemento de um `Array` chamado `imagens:`

```
imagens[1].width
```

Os seus elementos também não precisam ser todos do mesmo tipo de dado, como é necessário em outras linguagens tipadas como Java ou C. Assim, o `Array` abaixo é perfeitamente válido:

```
var a= new Array();
a[0]="JavaScript";
a[1]=true;      //booleano
a[2]=4.6;
a[3]=valores;  //Array do exemplo acima.
```

Também é possível inicializar *arrays* através da atribuição direta de todos os seus valores como parâmetros do construtor. As linhas abaixo declarariam e inicializariam os valores de `vetor` como as anteriores:

```
vetor=new Array(34,23,1,45,29,10);
var vetor=[34,23,1,45,29,10];
```

Nesta sintaxe, posições indefinidas podem ser incluídas, simplesmente omitindo os valores entre as vírgulas. Por exemplo, tem-se abaixo um *array* de 5 elementos, mas apenas os extremos são definidos na criação:

```
var cheioDeVazios=[1, , , ,5];
cheioDeVazios[9]=0;//agora o 9 é definido
```

Esta forma também pode ter expressões não se restringindo a valores constantes:

```
var base=1024;
var ConjuntoBase=[base,base+60,base/100];
```

Os *arrays* em JavaScript não têm seus índices armazenados em áreas consecutivas. Na realidade, a memória é *alocada*

apenas para os elementos com valores. Assim, quando você fornece as linhas de código que seguem , só são armazenadas as posições 0 e 100, e não são utilizadas as 99 posições de memória entre eles:

```
b[0]=1;
b[100]="elemento cem";
```

A forma de criar um *array* multidimensional em JavaScript é criar um *array de arrays*, que pode ser criado e inicializado pela seguinte sintaxe:

```
var matriz=[ [1,2,3],[4,5,6],[7,8,9]];
```

Os *arrays* têm apenas uma propriedade: `length`, que especifica quantos elementos ele tem. Como os índices sempre começam de zero, o valor de `length` será uma unidade maior que o índice do último elemento. Esta propriedade diferente da propriedade semelhante dos objetos `String` pode ser alterada (diz-se que é *read/write*: para leitura e escrita).

Se você estabelecer para `length` um valor maior que o real, novos elementos com valores indefinidos serão adicionados ao final do *array* para aumentá-lo até o índice `length-1`. Se você especificar um novo valor para ela, menor que o número de elementos existentes, todos os valores excedentes serão descartados. Diz-se que você estará **truncando** o `Array`. Truncar um *array* é a maneira possível em JavaScript de remover seus elementos.

14.1 - Propriedade do objeto embutido `Array`

Propriedade	Significado
`length`	Tamanho do `Array`, número do último índice do *array* mais `1`

Os *arrays* podem ser manipulados através dos diversos métodos descritos na tabela 14.2.

14.2 - Propriedade do objeto embutido `Array`

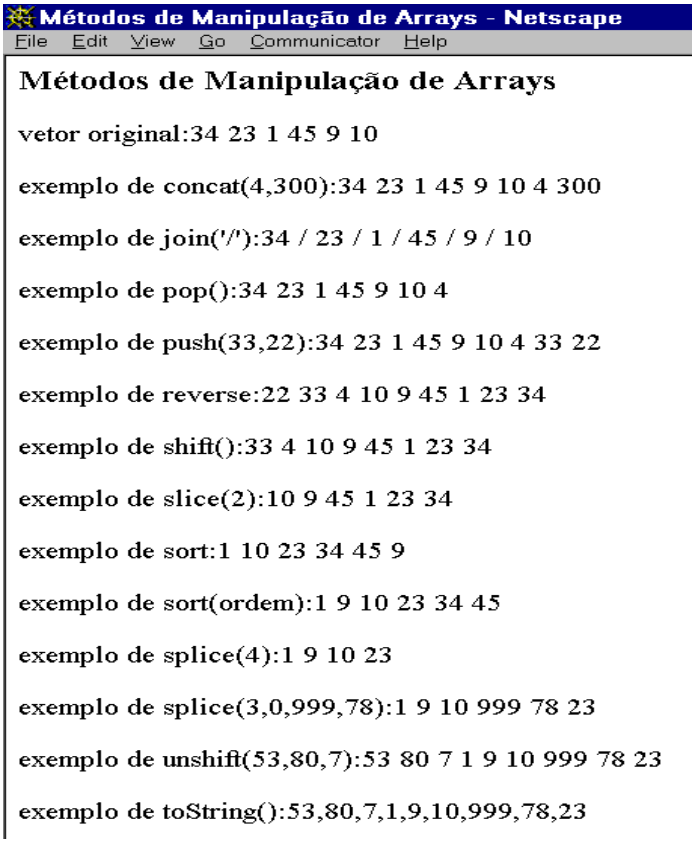
Métodos	Significado
`concat(v,..)`	Concatena os `v` com o *array*, cria um novo *array* não modificando o *array* original.
`join(s)`	Retorna uma *string* através de junção de cada elemento do *array* usando a *string* `s` como separador de palavras. Se `s` for omitido usará vírgula como separador.
`pop()`	Deleta o último elemento do *array* e decrementa a propriedade `lenght`. Junto com `push()` possibilita implementar o controle de dados como *pilhas*.
`push(v,...)`	Adiciona os valores `v` ao final do *array* , modificando a propriedade `lenght`.
`reverse()`	Inverte a ordem dos elementos do *array* sem criar que um novo *array* seja criado.
`shift()`	Remove o primeiro elemento do *array*, mudando o *índice* de todos os elementos subseqüentes. Não cria um novo *array*.
`slice(i,f)`	Retorna um *sub-array* contendo do elemento `i` ao `f`, mas sem incluir o elemento `f`. Se `f` não for fixado retorna todos os elementos de `i` ao final do *array*. O *array* não é modificado.
`sort(funord)`	Ordena um *Array*, sem copiá-lo. Se for usada sem argumento, os elementos são convertidos para *string* e arranjados em ordem alfabética. O argumento deve ser uma função de ordenação.
`splice(i,n,v....)`	Remove `n` elementos de um *array*, a partir de `i`, ou insere os `v` elementos a partir de `i`. O *array* é modificado. Se o valor `n` não tiver sido especificado, todos a partir de `i` serão removidos.
`toString()`	Converte um *array* para *string*, separando cada elemento por vírgulas
`unshift(v..)`	Insere `v` valores no início do *array*, mudando o índice dos demais.

Para entender exatamente o que cada método faz e como usá-lo, observe sua descrição na tabela, depois veja como ele é empregado no trecho de código que segue e observe o resultado obtido com sua execução na figura 14.1.

```
<HTML>
 <HEAD>
 <TITLE>Métodos de Manipulação de
Arrays</TITLE>
 </HEAD>
 <BODY>
 <H3>
 Métodos de Manipulação de Arrays
 </H3>
 <SCRIPT LANGUAGE="javascript">
 //exemplificando a inicializacao
    vetor = new Array(6);
    vetor[0] = 34;
    vetor[1] = 23;
    vetor[2] =  1;
    vetor[3] = 45;
    vetor[4] =  9;
    vetor[5] = 10;
    Vetor=new Array(34,23,1,45,9,10);
    var VetorCopy=[34,23,1,45,9,10];
    var base=1024;
    var vetorBase=[base, base+1, base+10,
    base+100, base/2, base%300];
    document.write("vetor original:");
    for (i = 0; i < vetor.length; i++)
      {document.write(vetor[i]+" " );}

    //exemplo do metodo
    //array.concat(valores,...)
    document.write("<br><br>exemplo de"
    +"concat(4,300):");
    Vetor=Vetor.concat(4,300);
    for (i = 0; i < Vetor.length; i++)
      {document.write(Vetor[i]+" " );}
    //exemplo de array.join(separador);
    document.write("<br><br>exemplo de"+
    "join('/'):"+vetor.join(" / "));

    //exemplo do metodo array.pop();
    document.write("<br><br>exemplo de"
    +"pop():");
    Vetor.pop();
    for (i = 0; i < Vetor.length; i++)
      { document.write(Vetor[i]+" " ); }
    //exemplo de array.push(valor,...);
    document.write("<br><br>exemplo de"+
    "push(33,22):");
    Vetor.push(33,22);
    for (i = 0; i < Vetor.length; i++)
      { document.write(Vetor[i]+" " );}
```

```
    Vetor.reverse();
    document.write("<br><br>exemplo de"
    +"reverse:");
    for (i = 0; i < Vetor.length; i++)
      {  document.write(Vetor[i]+" "); }
    Vetor.shift();
    document.write("<br><br>exemplo de"+
    "shift():");
    for (i = 0; i < Vetor.length; i++)
      {document.write(Vetor[i]+" "); }
    Vetor=Vetor.slice(2);
    document.write("<br><br>exemplo de"
    +"slice(2):");
    for (i = 0; i < Vetor.length; i++)
      { document.write(Vetor[i]+" ");}
    Vetor.sort();
    document.write("<br><br>exemplo de"
    +"sort:");
    for (i = 0; i < Vetor.length; i++)
     {  document.write(Vetor[i]+" " ); }
    function ordemNumerica(a,b)
    {return a-b;}//define uma função
    Vetor.sort(ordemNumerica);
    document.write("<br><br>exemplo de"
    +"sort(ordem):");
    for (i = 0; i < Vetor.length; i++)
     { document.write(Vetor[i]+" "); }
    Vetor.splice(4);
    document.write("<br><br>exemplo de"
    +"splice(4):");
    for (i = 0; i < Vetor.length; i++)
      { document.write(Vetor[i]+" "); }
    Vetor.splice(3,0,999,78);
    document.write("<br><br>exemplo de"
    +"splice(3,0,999,78):");
    for (i = 0; i < Vetor.length; i++)
      { document.write(Vetor[i]+" "); }
    Vetor.unshift(53,80,7);
    document.write("<br><br>exemplo de"
    +"unshift(53,80,7):");
    for (i = 0; i < Vetor.length; i++)
      { document.write(Vetor[i]+" "); }
    Vetor.toString();
    document.write("<br><br>exemplo de"
    +"toString():"+Vetor);
    </SCRIPT>
  </BODY>
</HTML>
```

Figura 14.1- Tela gerado usando o código que exemplifica os elementos da tabela 14.2



2. O Objeto `Math`

Este objeto é utilizado para armazenar constantes úteis a operações matemáticas e funções para efetuar diversos tipos de cálculo. As constantes e funções de `Math` podem ser invocadas usando expressões como as dos exemplos abaixo:

```
Math.floor(1.999)//retorna o valor 1
Math.floor(-1.01)//retorna o valor -2
Math.ceil(1.0001)//retorna 2
Math.ceil(-1.99)//retorna -1
Math.round(2.5)//retorna 3
Math.rount(-2.5)//retorna -2
radianos = graus * Math.PI / 180;
numero = Math.ceil(Math.random()*100)-1;
x= -b + Math.sqrt(Math.pow(b,2) - 4*a*c);
```

`Math` não é uma classe de objetos como `String`, mas um objeto que contém referência a funções e constantes matemáticas, não operando realmente nos objetos como os métodos. Para conhecê-las e entender o que cada uma faz, observe suas descrições nas tabelas 14.3 e 14.4.

14.3 - Constantes do objeto embutido `Math`.

Constante	Significado
`E`	*e*, base dos logaritmos naturais (≈2.71828..).
`LN10`	*ln10*, logaritmo natural de 10 ((≈2.302585...)
`LN2`	*ln2,* logaritmo natural de 2 ((≈0.69314718...)
`LOG10E`	*log e,* logaritmo da constante *e* na base 10 (≈0.434294...)
`LOG2E`	log_2 *e,* logaritmo na base 2 ca constante *e* ((≈1.442695...)
`PI`	Valor da constante π (≈3.1415...)
`SQRT1_2`	Valor do inverso da raiz quadrada de 2: 1/√2 ((≈0.707...)
`SQRT2`	Valor da raiz quadrada de 2: √2 (√1.4142..)

14.4 - Funções do objeto embutido `Math`.

A função **`atan2(y,x)`** é útil na conversão entre coordenadas polares e cartesianas.

Ela calcula o angulo θ do par cartesiano `(x,y)`. É chamada de **`atan2`**, pois introduz uma outra forma de calcular o arco tangente, agora, fornecendo 2 argumentos.

Repare na ordem dos argumentos na função: o `y` (que é usado como numerador no cálculo da tangente) é introduzido primeiro!

Funções	Significado
`abs(x)`	Valor absoluto (sem sinal + ou -) de qualquer número `x`.
`acos(x)`	Arco coseno, ou função inversa do coseno de `x`, o parâmetro `x` deve ser um valor entre -1.0 e 1.0. O resultado é um valor entre -π/2 e π/2.
`asin(x)`	Arco seno de `x`, ou função inversa do seno de `x`, o parâmetro `x` deve ser um valor entre -1.0 e 1.0. O resultado é um valor entre -π/2 e π/2.
`atan(x)`	Arco tangente de `x`, ou função inversa da tangente de `x`. O resultado é um valor entre -π/2 e π/2.
`atan2 (y,x)`	Ângulo no sentido contrário aos ponteiros do relógio entre o eixo *X* positivo e o ponto de coordenadas `(x,y)`.
`ceil(n)`	Retorna o um número inteiro mais próximo, que seja igual ou maior que `n`. Quando `n` for negativo retorna um valor menos negativo que `n`, ou seja maior e mais próximo ao zero que `n`.
`cos(a)`	Retorna um valor entre -1 e 1, que é o cosseno do ângulo `a` (o parâmetro `a` deve ser fornecido em radianos).
`exp(n)`	Eleva a constante *e* a potência `n`, ou seja calcula: e^n.
`floor (n)`	Retorna o número inteiro mais próximo, que seja igual ou menor que `n`. Quando `n` for negativo, retorna um valor mais negativo que `n`, e não mais próximo de zero.

14.4 - Funções do objeto embutido `Math`.(continuação)

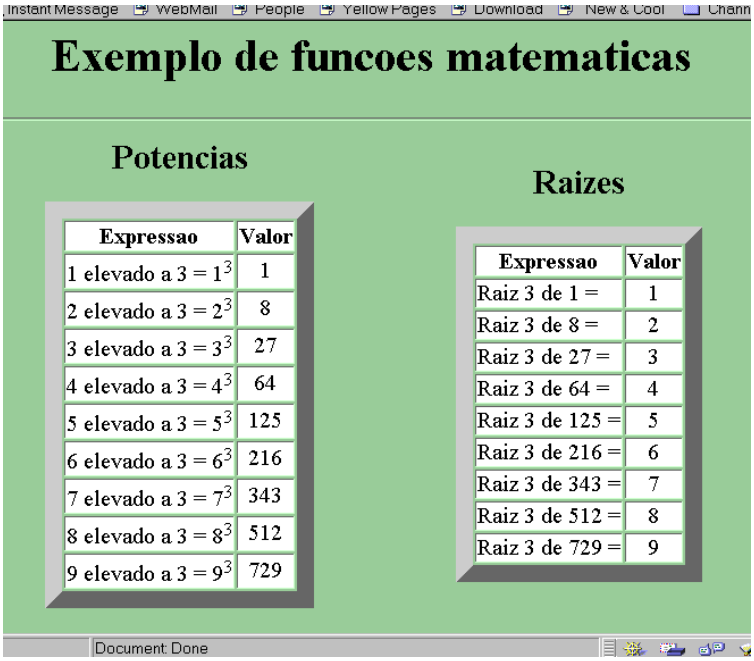
Funções	Significado
`log(n)`	Logaritmo natural de `n`: log_e `n`. O parâmetro deve ser um número maior que zero.
`max(a,b)`	Retorna o maior entre dois números `a` e `b`.
`min(a,b)`	Retorna o menor entre dois números `a` e `b`.
`pow(x,y)`	Retorna `x` a potência `y`: x^y. Se o resultado for um número complexo ou imaginário o resultado será `NaN`.
`random()`	Gerador de números *pseudo-aleatórios*, ou *pseudo-randômicos* entre 0.0 e 1.0.
`round(n)`	Arredonda `n` para o valor inteiro mais próximo.
`sin(a)`	Seno do ângulo `a` (em radianos).
`sqrt(n)`	Raiz quadrada de um número `n`. O parâmetro `n` deve ser positivo ou zero. Para valores negativos o valor de retorno será `NaN`.
`tan(a)`	Tangente do ângulo `a` (em radianos).

O código abaixo exemplifica o uso de algumas destas funções matemáticas .

```
<html>
 <head>
   <title>
   Exemplo de constantes matematicas
   </title>
   <script language="JavaScript">
   <!--
   function criaTabela()
   {
   document.write("<table border=15
   bgcolor=white>");
   document.write("<tr align=center
   height=110>");
   document.write("<th>Expressao</th>");
   document.write("<th>Valor</th>");
   document.write("</tr>");
   }
   function fechaTabela()
   {  document.write("</table>");  }
   //-->
   </script>
 </head>
 <body bgcolor=darkseagreen>
  <h1>
  Exemplo de funcoes matematicas</h1>
  <hr>
  <table width=100%>
  <tr align=center>
  <td>
  <h2>Potencias</h2>
  <script language="JavaScript">
```

```
  <!--
  criaTabela();
  text="3";
  for (lin = 1; lin < 10; lin++)
  {
  document.write("<tr align=center>\n");
  document.write("<td>",lin," elevado a 3
  = ",lin,text.sup(),"</td>");
  document.write("<td>", Math.pow(lin,3),
  "</td>");
  document.write("\n</tr>\n");
  }
  fechaTabela();
  //-->
  </script>
  </td>
   <td>
   <h2>Raizes</h2>
    <script language="JavaScript">
    <!--
    criaTabela();
    var num;
    for (lin = 1; lin <10; lin++)
    {
    num=Math.pow(lin,3)
    document.write("<tr height=100
    align=left>\n");
    document.write("<td>Raiz 3 de ",num,"
    =</td>");
    document.write("<td align=center>",
    Math.round(Math.pow(num,1/3)),
    "</td>");
    document.write("\n</tr>\n");
    }
    fechaTabela();
    //-->
    </script>
    </td>
    </tr>
    </table>
  </body>
</html>
```

Figura 14.2 - Tela gerada usando o código que exemplifica os elementos da tabela 14.4



3. O Objeto `Date`

O objeto `Date` permite manipular datas e horários, como por exemplo verificar a data atual ou determinar a diferença entre duas datas. Este objeto precisa ser criado através do operador new. Há 4 formas de criar uma data:

G.M.T. abreviatura de Greenwich Mean Time (hora média de Greenwich).

- `new Date();`
- `new Date(milisegundos);`
- `new Date(datastring);`
- `new Date(ano,mes,dia,hora,min,seg,ms).`

Na primeira forma, sem parâmetros, o construtor `Date()` cria um objeto e atribui a ele a data atual do sistema:

```
dataHoje = new Date();
```

Quando um valor numérico é passado como parâmetro, a data armazenada é obtida a partir do número de milisegundos entre a data atual e a hora zero (meia-noite) de GMT em 1º. janeiro de 1970:

```
new Date(5000)//cria uma data que
     //representa 5 segundos de 1/1/1970.
```

Na terceira forma, uma *string* com a data, cujo horário é opcional, é passada como parâmetro. Esta *string* é geralmente

dependente do ambiente, devendo ser uma das formas aceitas para datas no formato IETF. Esta é uma padronização para uso em *e-mails* e outras comunicações na Internet. Pode ser que você já tenha observado esta *string* de datas, nas quais os valores se parecem com:

```
"Wed,15 Feb 2007 17:41:46 - 0400"
"February 15,2007 17: 41:46"
"1 15, 2007 17:41:46"
```

Na última forma, se forem passados de 2 a 7 parâmetros, é suposto que o tempo está sendo especificado usando o tempo local, não o tempo universal UCT - *Universal Coordinate Time* ou GMT. Podem ser passados até 7 valores, pois todos, menos o campo correspondente ao ano e mês, são opcionais. Assim, a data pode ser estabelecida através do ano, mês e dia:

```
data = new Date(2007, 1, 15);
```

ou através do ano, mês, dia, hora, minuto, segundo, milisegundos:

```
data = new Date(2007,1,15,17,41,46,400);
```

Neste último formato, o ano deve se apresentar em 4 dígitos, os meses como um número inteiro entre 0 (janeiro) e 11 (dezembro), os dias do mês como inteiros entre 1 e 31, as horas em inteiros de 0 (meia-noite) a 23, os minutos e segundos em inteiros de 0 a 59, e os milésimos de segundos em inteiros de 0 a 999.

O objeto `Date` não tem nenhuma propriedade, mas tem uma série de métodos que permitem a manipulação de cada parte de uma data em separado. Depois do objeto `Date` ter sido criado usando qualquer uma das 4 formas acima, pode ser usado com os métodos descritos na tabela 14.5. A maioria destes têm a mesma função em duas formas, uma para obter a data e o tempo e outra para defini-los. No primeiro caso, os métodos iniciam com `get` (obtenção) e no segundo com `set` (atribuição).

14.5 - Métodos do objeto embutido `Date`

Métodos	Significado
`getDate()/ setDate()`	Obtém ou define o dia do mês (de 1 a 31).
`getDay()`	Obtém o dia da semana (de 0 a 6).
`getFullYear()/ setFullYear()`	Obtém ou define o ano com 4 dígitos.
`getHours()/ setHours()`	Obtém ou define a hora de um objeto Date (de 0 a 23).
`getMilliseconds() setMilliseconds()`	Obtém ou define o campo milissegundos (hora-local).
`getMinutes()/ setMinutes()`	Obtém ou define o minuto (entre 0 e 59).
`getMonth()/ setMonth()`	Obtém ou define o mês (de 0 a 11).
`getSeconds()/ setSeconds()`	Obtém ou define os segundos (entre 0 e 59).
`getTime()/ setTime()`	Obtém ou define o nº de milissegundos desde 01/01/70.
`getTimezoneOffset()`	Obtém o fuso horário em minutos entre a hora local e GMT.
`getUTCDate()/ setUTCDate()`	Obtém ou define o dia do mês (de 1 a 31) quando `Date` estiver em UTC
`getUTCDay()`	Obtém o dia da semana (de 0 a 6) quando `Date` estiver no tempo universal ou UTC
`getUTCFullYear() setUTCFullYear()`	Obtém ou define o ano com 4 dígitos quando `Date` estiver no tempo em UTC.
`getUTCHours()/ setUTCHours()`	Obtém ou define a hora (de 0 a 23) de um objeto `Date` em UTC.
`getUTCMilliseconds() setUTCMilliseconds()`	Obtém ou define o campo milissegundos quando `Date` estiver em UTC.
`getUTCMinutes() setUTCMinutes()`	Obtém ou define o campo minutos em UTC.
`getUTCMonth() setUTCMonth()`	Obtém ou define o campo milissegundos quando `Date` estiver em UTC.

14.5 - Métodos do objeto embutido `Date`. (continuação)

Métodos	Significado
`getUTCSeconds()` `setUTCSeconds()`	Obtém ou define o campo segundos em UTC.
`getYear()/` `setYear(year),year pode ter 2 ou 4 digitos, se tiver 2 os primeiros serão 19`	Obtém ou define o ano em 2 dígitos. Esta em desuso depois do ano 2000, melhor usar as versões `FullYear`.
`parse(datastring)`	Calcula o tempo em milissegundos entre a `datastring` fornecida e a zero hora de 1/1/1970.
`toGMTString()`	Converte uma data para o formato `datastring` usando o horário GMT.
`toLocaleString()`	Converte uma data para o formato `datastring` usando o fuso horário local.
`toString()`	Retorna `Date` como uma *string* legível no fuso local.
`toUTCString()`	Retorna o objeto Date convertida em uma *string* legível em UTC.
`UTC(ano,mes,dia,hora,min,seg,ms)`	Converte uma data para milissegundos desde o tempo zero: 1/jan/1970 0:0:0 .
`valueOf()`	Representação numérica da data. Mesmo número de milissegundos de `getTime .`

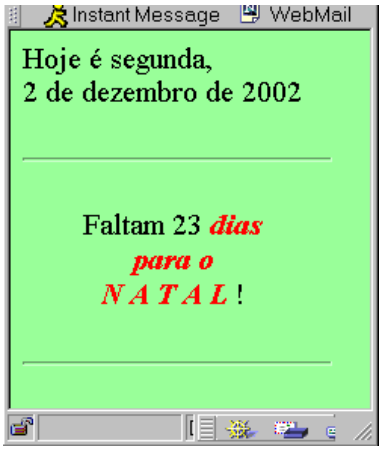
O código abaixo exemplifica o uso de alguns destes métodos.

```
<HTML>
  <HEAD>
    <TITLE>Usando o objeto Data</TITLE>
  </HEAD>
  <BODY BGCOLOR=lightgreen>
    <SCRIPT language="Javascript">
    mes = new Array(12);
    mes[0]  = "janeiro";
    mes[1]  = "fevereiro";
    mes[2]  = "março";
    mes[3]  = "abril";
    mes[4]  = "maio";
    mes[5]  = "junho";
    mes[6]  = "julho";
    mes[7]  = "agosto";
    mes[8]  = "setembro";
    mes[9]  = "outubro";
    mes[10] = "novembro";
    mes[11] = "dezembro";
```

```
    semana = new Array("domingo",
    "segunda", "terça", "quarta",
    "quinta", "sexta", "sábado");

    var hoje = new Date();
    //Obtém a data de hoje
    var AnoAtual=hoje.getFullYear();
    var MesAtual=hoje.getMonth();
    var DiaAtual=hoje.getDate();
    document.write("<p>Hoje é " +
    semana[hoje.getDay()] + ",<br>" +
    DiaAtual+" de " + mes[MesAtual] +
    " de " + AnoAtual+"</p><hr>");

    //  O mes é de 0 a 11
    var AnoNatal=AnoAtual;
    // vejo se ja passou o Natal no ano
    if (MesAtual==11){
         if (DiaAtual>25) {AnoNatal++;}};
    //dado para o proximo Natal!
    var data = new Date(AnoNatal,11,25);
    //  Número de segundos por dia
    var nsPorDia = 24*60*60*1000;
    var ndias=(data.getTime()-
              hoje.getTime()) / nsPorDia;
    ndias = Math.round(ndias);
    natal=new String(" dias <br> para o
    <br> N A T A L");

    document.write("<center><p>Faltam " +
    ndias+natal.bold().fontsize("+").
    italics().fontcolor("red")+" !
    </center> </p> <hr> ");
    </SCRIPT>
  </BODY>
</HTML>
```

Este código gera a página mostrada na figura 14.3.

Figura 14.3 - Tela gerada usando o código que exemplifica os elementos da tabela 14.5.



Para entender exatamente o que cada método de `Date` faz e como usá-lo, observe sua descrição na tabela 14.5, depois empregue-o no trecho de código acima e utilize `document.write` para observar o resultado obtido com sua execução. Por exemplo, para ver como é a `datastring` gerada pelo método `toString()`. Inclua as linhas seguintes no final do código anterior antes de `</SCRIPT>`:

```
document.write(" <p> Hoje é: " +
     hoje.toString() + "</p> <hr> ");
```

Este é o último dos 4 objetos predefinidos ou embutidos da linguagem. Como falamos na aula anterior, há ainda outros tipos de objetos em JavaScript. Um destes são os objetos do *browser* que veremos na próxima aula.

Exercícios:

1. Utilize a estrutura do exemplo de `Array` para testar todas as outras formas de inicialização apresentadas. Isto é, bata o exemplo até o trecho onde é exibido o `Array` original, ou seja até:

```
     //exemplo do metodo
     //array.concat(valores,...)
```

e rode o código verificando os valores do outros *arrays* do exemplo:.

```
Vetor=new Array(34,23,1,45,9,10);
VetorCopy=[34,23,1,45,9,10];
vetorBase.....
```

Visualize em cada caso os resultados que terá em um navegador.

2. Inclua o restante do código do exemplo `Array` e o rode com cada um dos novos *arrays*. Tente prever em cada um dos casos o que cada método de *array* estará escrevendo na tela do seu navegador.

3. Utilize a estrutura do exemplo de funções matemáticas para testar as outras formas de cálculo. Por exemplo, faça uma tabela de logaritmos e de cosenos de diversos ângulos.
Visualize em cada caso os resultados que terá abrindo uma página com o código em um navegador.

4. Inclua a possibilidade de cronometrar o tempo no código que exemplificou o uso de funções, empregando os métodos do objeto `Date`. Cronometre também o tempo que é gasto para gerar suas tabelas do exercício 1 (basta fazer a diferença de tempo do sistema no início e no fim).

Resumo:

Nesta aula você conheceu em detalhes as características dos objetos predefinidos `String`, `Math` e `Date` da linguagem JavaScript. Na próxima aula continuaremos neste assunto estudando os objetos do *browser*.

Auto-avaliação:

Você concluiu com facilidade os exercícios e entendeu as diversas propriedades dos objetos novos? Se o uso de alguma propriedade não ficou muito claro, não se preocupe, você sempre pode procurar as tabelas e ver como funcionam antes de usá-las em um programa! O mais importante, por enquanto, é saber que eles existem. Continuaremos vendo objetos e o assunto da próxima aula é realmente muito importante para um bom programador em *client-side* JavaScript: como funciona cada janela do navegador ou cada *frame* dentro da janela.