Javascript

P. Kantek

20 de abril de 2024

Sumário

1 Javascript e código embutido

Em se tratando de uma página WEB pode-se pensar no seguinte modelo conceitual

funcionalidade	responsável	onde atua
conteúdo da página	HTML	interpretada no cliente
design	CSS	interpretada no cliente
programação servidor	PHP, Asp, Java, Ruby...	servidor
programação cliente	Javascript	cliente

Cedo na história da Internet sentiu-se falta de uma ferramenta que permitisse algum grau de manipulação das informações no plano da máquina cliente. Era o mínimo que se precisava para garantir algum tipo de interatividade às páginas web.

Javascript foi criada pela Netscape em parceria com a Sun Microsystems. A primeira versão Javascript 1.0 foi lançada em 1995 e implementada em março de 1996 no Netscape Navigator 2.0. Logo a seguir começou a guerra dos browsers e a Microsoft em resposta lançou a linguagem Jscript baseada no Visual Basic junto com o IE 3.0. Ambas tem como característica serem processadas (interpretadas e executadas) pelo browser usando recursos exclusivamente da máquina cliente. Tanto a Microsoft como a Netscape acabaram lançando versões Javascript para rodarem no servidor, mas estas não nos interessam aqui.

Por ser uma linguagem bem abrangente, Javascript acabou caindo também no que eu chamo de maldição da criação de universos[1]. Muitos programadores resolveram usar Javascript para que ela (a Javascript) gerasse o HTML correspondente. Péssimo negócio responsável em parte pela má fama que a linguagem andou tendo. Deve-se limitar a Javascript a manipular dados das páginas e nunca a substituir uma correta marcação HTML previamente estabelecida.

Outro aspecto a ser evitado é usar o Javascript para constranger de alguma maneira o usuário. Todos aqui já devem ter alguma vez entrado em site e:

- perceber que a parte superior do navegador sumiu, impedindo fechar a janela
- a cada click em um link ver que uma janela popup de propaganda se abre
- ao fechar a janela do navegador perceber um monte de janelas abertas não solicitadas
- não deixar você sair
- entre outras ações mais ou menos condenáveis

Esta é outra das razões da má fama da Javascript. Daqui pode-se tirar a recomendação de não manipular a janela do navegador do usuário pois é ele quem decide se quer abrir, fechar, redimensionar, voltar, sair, enfim, fazer o que bem entende.

Um aspecto muito instigante é que Javascript pode ser usada em conjunto com outras linguagens para distribuir o fluxo da programação nos dois extremos do palco onde a computação ocorre.

De um modo geral usar javascript em conformidade com os padrões web em geral implica em obdecer aos seguintes princípios:

Usar javascript não obstrutivo significa que o conteúdo da página deve estar presente e funcional mesmo que o usuário use um ambiente sem javascript, mesmo que neste caso se perca parcela de usabilidade.

Usar Javascript visando apenas incrementar a usabilidade da página significa deixar de fora efeitos espetaculares mas sem qualquer utilidade: figuras acompanhando o cursor, objetos voando pela página, luzes piscando...

Escrever scripts em arquivos externos para serem agregados ao documento e portanto não escrever javascript na marcação HTML. Este princípio sugere manter Javascript na camada de comportamento, deixando a camada de estruturação (HTML) e a camada de apresentação (CSS).

A Javascript tem como nome oficial ECMAScript (European Computer Manufacturers Association) que lançou a norma de Javascript em 1997. A mesma norma foi reconhecida pela ISO em 1998. É uma linguagem que suporta a orientação a objetos, embora deixe de cumprir alguns requisitos do modelo conceitual. A OOP é um paradigma de programação no qual os elementos da programação simulam objetos. Um objeto neste universo possui atributos ou propriedades e métodos. Os atributos são os elementos passivos (dados) enquanto os métodos são os ativos (programas). Fazendo uma analogia de um objeto com uma pessoa, as suas propriedades poderiam ser nome, seco, altura, naturalidade,

[1] Eu costumo brincar: enquanto o engenheiro civil precisa respeitar (e obedecer) à Lei da Gravidade, enquanto o médico não pode deixar a temperatura de um paciente ultrapassar 41^o e enquanto um advogado precisa se entender com o código penal, um analista de sistemas não sofre nenhuma restrição: atua como criador de universos. Não há regras prévias, e ele é um autêntico *deus ex machina*. Isto é uma delícia, mas pode ser muito perigoso. Principalmente quando o analista resolve usar a também muito perigosa estratégia *olhe mamãe, agora sem as mãos.*

profissão, etc. Os métodos são as ações que podem ser executadas pelo objeto. No nosso exemplo poderiam ser comer, andar, dormir. Já que os métodos em outros ambientes são conhecidos como funções eles serão escritos aqui como comer(), andar() e dormir().

Os objetos em javascript podem ser de 3 tipos:

- internos da linguagem, pré-definidos e que o programador apenas usa. Por exemplo, datas, strings e arrays.
- do ambiente de hospedagem (por exemplo, o navegador) como window e document.
- do desenvolvedor criados e usados por ele.

1.0.1 Caixa de diálogo

É uma janela pop-up que é usada pelo javascript para comunicar informações ou apanhar um dado. O primeiro tipo chamado caixa de alerta. O alerta vai ser feito no objeto Window que é por onde o javascript conversa com o usuário. A maneira de chamar a caixa é

```
window.alert("mensagem");
```

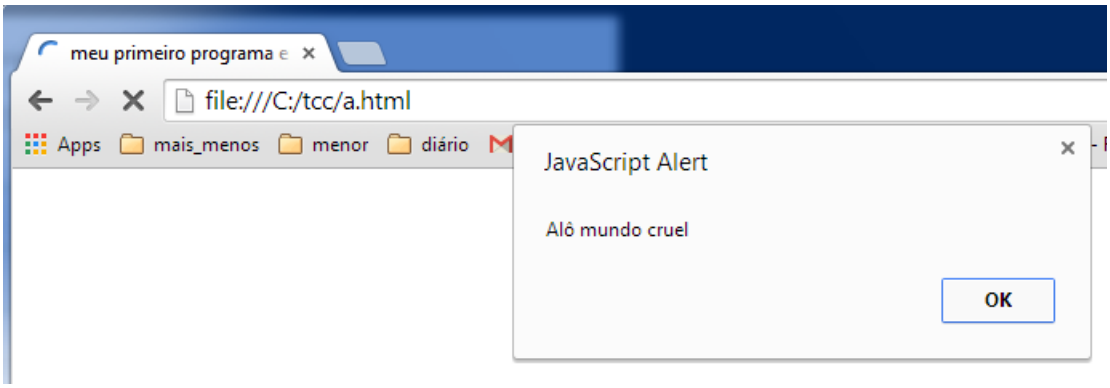
No caso particular do objeto `window` que é o objeto de mais alto nivel, este nome pode ser omitido e o comando pode ser

```
alert("mensagem");
```

Eis a seguir o nosso primeiro programa de javascript, chamado como em todas as linguagens `alomundo`.

```
<head>
  <title> meu primeiro programa em javascript </title>
  <meta charset=utf-8">
  <script type="text/javascript">
    alert("Alô mundo cruel");
  </script>
</head>
<body>
</body>
</html>
```

Este programa foi digitado em um editor com capacidade de utf-8 e salvo com o nome `c:/tcc/a.html`. Dai, chamando-se o browser obteve-se o seguinte resultado



Note-se que esta é a janela alert do Chrome. As do Firefox, Safari, Opera e IE são ligeiramente parecidas, mas todas com a mesma funcionalidade. Note que para inserir código javascript dentro da página HTML é necessária uma tag `script` com seu atributo `type`. A síntaxe preve a abertura (`<script>`) e o seu encerramento (`</script>`) e dentro disso o código javascript. Neste script há um método nativo da linguagem que pertence ao objeto window denominado `alert()` que espera uma mensagem como parâmetro. É esta mensagem que será escrita na caixa de alerta e deverá estar sempre entre aspas dupla ou entre aspas simples. A linha da declaração da função termina por um ";". Note que a definição do programa javascript está dentro do cabeçalho do arquivo (entre `<head>` e `</head>`). Poderia estar também entre `<body>` e `</body>`. Iria funcionar igual, mas aí estaríamos desrespeitando a separação das camadas de desenvolvimento, misturando comandos HTML com comandos javascript.

Se você quiser mostrar uma mensagem em duas linhas, basta incluir um caracter de controle ASCII dentro da mensagem. Eis como ficaria

```
<script type="text/javascript">
  alert("Alô mundo \n cruel");
</script>
```

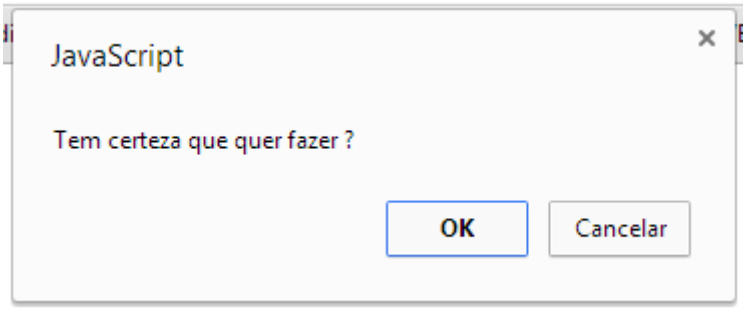
A propósito eis uma tabela dos marcadores ASCII

`\b`	back space
`\f`	form feed
`\n`	new line
`\t`	tabulação
`\v`	tabulação vertical
`\"`	aspas duplas
`\'`	aspa simples
`\0`	zero decimal (nulo)
`\\`	uma barra invertida
`\a`	alerta (beep)
`\r`	return
`\0oo`	numero octal
`\xhh`	numero hexadecimal

O método `confirm` do objeto window, funciona como se fosse um `alert`, mas depois da mensagem, ele escreve dois botões denominados `OK` e `CANCELAR` e ao clicar em um deles, o usuário confirma ou cancela a operação. Veja o exemplo

```
<head>
  <title> meu primeiro programa em javascript </title>
  <meta charset=utf-8">
  <script type="text/javascript">
    confirm("Tem certeza que quer fazer ?");
  </script>
</head>
<body>
</body>
</html>
```

com este resultado:



2 Operadores em Javascript

Em Javascript uma expressão é tudo aquilo capaz de ser avaliado pelo interpretador da linguagem. Uma combinação de expressões com o uso de operadores também é uma expressão. Eis a seguir uma lista de operadores Javascript

operador	finalidade
+,-,*,/,%	adição, subtração, multiplicação, divisão real e resto
+	concatenação de strings
<,>, <=, >=	menor, maior, menor ou igual, maior ou igual
==, ! =	igualdade, não igual
&&, ||	E (∧) e OU (∨)
.	acesso a propriedade de um objeto
[]	indexar um array
()	chamar uma função
=	atribuir valor
new	criar um objeto

Um operando é uma variável ou um valor que junto com um operador forma uma expressão. O Javascript possui inúmeras regras de conversão no caso de haver misturas de tipos (numérico, alfanumérico, booleanos (`true` e `false`)). Note que Javascript possui uma qunatidade maior de operadores e um monte ainda maior de regras de conversão. Não é uma boa

idéia usar tais regras de maneira obsessiva, já que elas nem sempre coincidem com as outras linguagens que eventualmente terá que ser usada (Java, C, C++, etc).

3 Declarações e estruturas

Uma lista de declarações e comandos que são usados em Javascript.

3.1 var

declara explicitamente uma ou mais variáveis. Alguns exemplos

```
var y;
var z = 8;
var a, b, c;
var p = 2, q = 3, w = 9;
```

3.2 function

define uma função Javascript. A síntaxe do uso é

```
function nome-da-função ([arg1, arg2, ... argn]) {
   script da função;
};
```

Por exemplo, seja uma função que recebe 2 números e imprime a média deles.

```
function media (a, b) {
   var resu = (a + b) / 2;
   alert (resu)
};
```

Dentro da função existe o comando `return` que quando executado devolve algum resultado e ENCERRA a execução da função.

3.3 if

Permite estabelecer comandos condicionais. Veja o formato

```
if (expressão) {
   ... bloco de comandos A }
[else {
   ... bloco de comandos B }]
```

Se a expressão resulta verdadeira o bloco A é executado. Se o bloco `else` existe, e a expressão é falsa então o bloco B é executado.

3.4 switch

Permite escolher uma dentre várias alternativas. Veja o formato

```
switch(expressão) {
   case resultado1: script1;
      break;
   case resultado2: script2;
      break;
   default: script-padrão;
};
```

Tudo o que se faz com `switch` pode ser feito com `if`, razão pela qual muitos programadores passam a usar exclusivamente o `if`.

3.5 for

permite criar um número determinado de iterações. Veja

```
for(expressão-inicial; condição-de-fim; atualização-variável-controle) {
   script;
};
```

Veja o exemplo

```
var msg="";
for (var i=0; i<5; i++) {
   msg = msg + i + ", ";
};
alert(msg);
```

3.6 while

Permite criar um laço. A síntaxe é

```
while(expressão) {
   script;
};
```

Veja o mesmo caso feito com o for acima.

```
var msg="";
i=0;
while(i<5) {
   msg = msg + i + ", ";
   i++;
};
alert(msg);
```

3.7 do/while

```
do {
   script;
} while(expressão);
```

permite que o ciclo seja executado pelo menos uma vez. Veja o mesmo ciclo acima

```
var msg="", i=0;
do {
   msg = msg + i + ", ";
   i++
} while (i<5);
alert(msg);
```

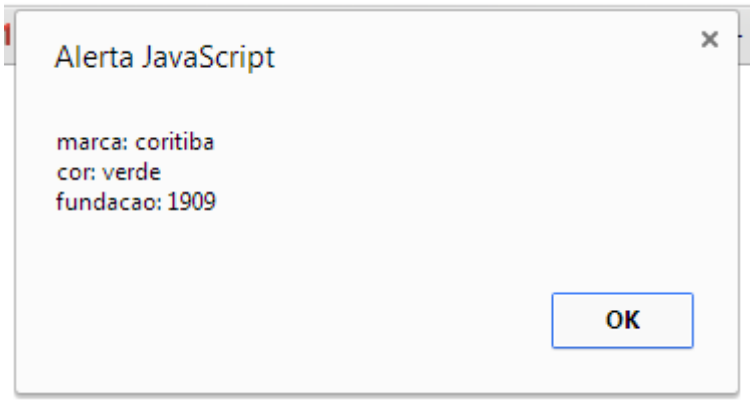
3.8 for/in

É usado para percorrer objetos ou arrays. Veja o exemplo

```
<head>
  <title> teste de for/in </title>
  <meta charset=utf-8">
  <script type="text/javascript">
  var time = {
     marca: "coritiba",
  cor: "verde",
  fundacao: 1909
  };
var msg="", k;
for (k in time){
   msg = msg + k +": "+time[k]+"\n";
}
alert(msg);
```

```
</script>
</head>
<body>
</body>
</html>
```

deu como resposta

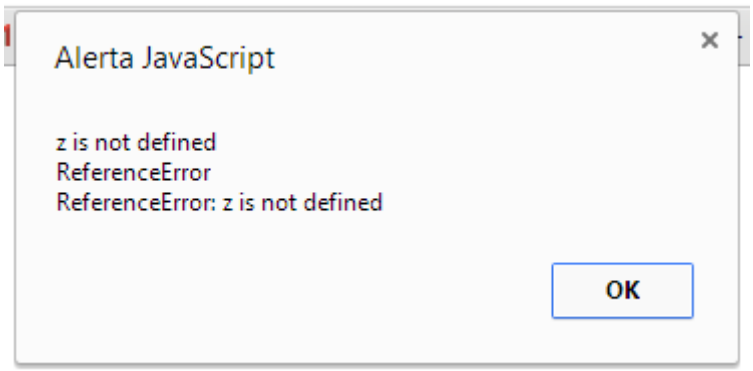


3.9 try/catch/finally

Estabelece um método para lidar com exceções tipicamente de erros. Eis um exemplo completo

```
try {
  var y = z + 1;
  alert(y);
  } catch(qual) {
     alert(qual.message + "\n" + qual.name + "\n" +qual.toString());
  };
```

que deu a seguinte mensagem de erro:



Como não há uma padronização no manuseio de erros é conveniente tomar a si esta tarefa. Veja como

```
var x = prompt("Informe um numero entre 5 e 20", "");
try {
   if (x<5){ throw ("erro1"); }
   if (x>20) { throw ("erro2"); }
   if (isNaN(parseInt(x))) { throw ("erro3"); }
}
catch(e) {
   if (e=="erro1") {alert("o numero deve ser maior do que 4");}
   if (e=="erro2") {alert("o numero deve ser menor do que 21");}
   if (e=="erro3") {alert("o numero deve estar entre 5 e 20");}
   }
```

4 Objetos

Um objeto em Javascript é uma coleção de dados. Tal coleção sempre existe na forma de pares nome/valor. Para criar um objeto usa-se o operador `new`. O modo de uso é

```
var nome-objeto = new construtor();
```

Existe um construtor genérico chamado `Object()` que cria um objeto genérico vazio. Para popular um objeto usa-se o operador `.` (ponto). Antes do ponto vem uma referência ao objeto, ou uma variável que representa o objeto. Depois do ponto o nome da propriedade do objeto. Depois vem a atribuição `=` e depois o conteúdo da propriedade. Veja o exemplo

```
var carro = new Object();
carro.marca = "Ford";
carro.dono = "Zeca";
carro.proximarevisao = function(anofabricacao, estadogeral, kilometragem){
   var quando;
   // script que calcula a proxima revisao em funcao do ano, do estado e da kilometragem
   return quando;
   }
carro.cor = azul;
```

Neste exemplo o objeto carro possui 3 propriedades e um método chamado `proximarevisao` que se destina a calcular a próxima revisao que o carro deve sofrer. A qualquer momento pode-se criar uma nova propriedade, bastanto definí-la. Pode-se também alterar uma propriedade já existente, bastando redefiní-la. Propriedades ou métodos novos ou alterados passam a existir somente após terem sido criados/alterados e seus efeitos não modificam valores anteriores no script. Há uma síntaxe alternativa para recuperar valores de objetos usando `[]` ao invés do `.`. Veja o exemplo

```
var donodocarro = carro[dono]; // equivale a var donodocarro = carro.dono;
```

Os objetos podem ser aninhados, ou seja, uma propriedade pode ser um outro objeto. Existe também uma síntaxe chamada literal, que consiste em definir a coleção de nomes/valor separando os nomes com vírgula e envolvenmdo a coleção com { e }. Veja o mesmo objeto acima definido

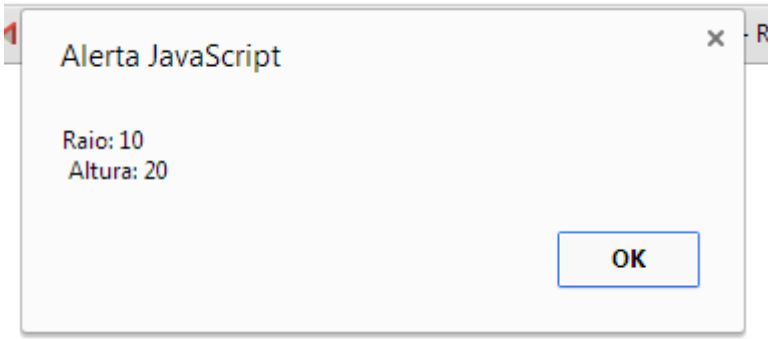
```
var carro = {
"marca": "Ford", "dono": "Zeca", "proximarevisao": function(anofabricacao, estadogeral, kilome
   var quando;
   // script que calcula a proxima revisao em funcao do ano, do estado e da kilometragem
   return quando;
   }, "cor": "azul"};
```

4.1 Construtor

É uma função capaz de criar objetos. Seja por exemplo uma necessidade de criar cilindros

```
function cilindro(r,h){
   this.raio = r;
   this.altura = h;
   };
   ...
   cilindroum = new cilindro(10,20);
   alert("Raio: " + cilindroum.raio + "\n Altura: "+cilindroum.altura);
```

que deu como resposta

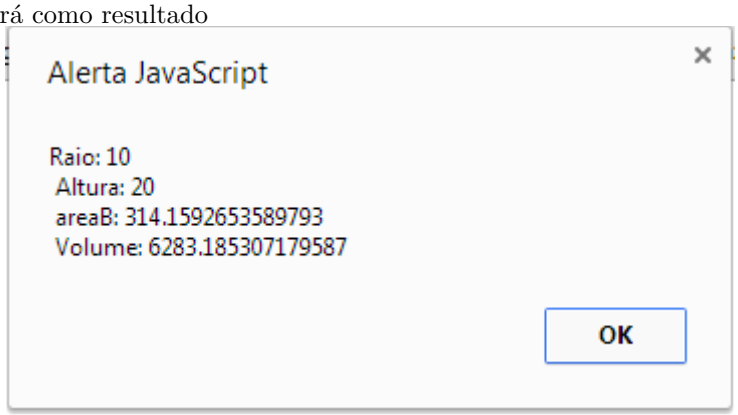


Note que a palavra `this` é uma referência ao objeto em questão dentro do qual ela esta sendo escrita. Só construir um construtor com os valores das variáveis não é grande coisa. Vai-se criar também algumas grandezas de um cilindro como por exemplo a área da base ($raio^2 \times \pi$) e o volume que é $base \times altura$ e fica

```
<html>
<head>
  <script type="text/javascript">
```

```
function cilindro(r,h){
   this.raio = r;
   this.altura = h;
   this.area = function calculoarea() {
      return Math.PI * Math.pow(this.raio, 2);
   };
   this.volume = function calculovolume() {
      return this.altura * this.area();
   };
   };
   cilindroum = new cilindro(10, 20);
   alert("Raio: " + cilindroum.raio + "\n Altura: "+cilindroum.altura
   + "\n areaB: "+cilindroum.area() + "\n Volume: "+ cilindroum.volume());
    </script>
</head>
<body>
</body>
</html>
```

Dará como resultado



5 Arrays

Uma coleção ordenada de dados, indexados sequencialmente a partir de zero. Para criar um array usa-se o construtor `Array()` precedido do operador `new`. Para popular um array passa-se uma lista de argumentos ao construtor como no exemplo

```
var brinquedos = new Array("autorama", "bobesponja", 33, "ferrorama");
```

A lista de elementos é ordenada. O primeiro é `autorama` e ele tem o índice zero. O último brinquedo é `ferrorama` que tem o índice 3. Para manipular um elemento de um array, escreve-se o nome do array e o número do índice entre colchetes. Assim,

```
brinquedos[0] // devolve a cadeia autorama
brinquedos[2] // devolve o número 33
...
```

Um array pode ter um elemento alterado, ou pode ter novos elementos sendo incluídos como em

```
brinquedos[0] = "barbie";
brinquedos[4] = "bonecasuzy";
```

E, um array pode conter um objeto, que neste caso deve ser fornecido entre { e }, como em

```
brinquedos[5] = {nome: "ken", preco: 200.00, compradoem: "lojasamericanas"};
```

Neste caso, para recuperar um atributo do objeto presente no índice 5 (o sexto) do array, a síntaxe é

```
brinquedos[5].preco // que vale 200.00
```

Finalmente, um elemento de um array pode ser ele também um array. Veja um exemplo completo

```
var sinf = ["nona", "inacabada", {nome: "novomundo", autor: "dvorak"}, [41, "mozart", true]];
alert(sinf[0]);  // responde nona
alert(sinf[2].autor); // responde dvorak
alert(sinf[3][1]); // responde mozart
```

Há ainda os chamados arrays associativos, nos quais os elementos são reconhecidos pelo seu nome e não pelo seu índice. Funcionam como se fossem objetos.

5.1 Propriedades e métodos do objeto array

Há 3 propriedades: `constructor`, `prototype` e `length`. Destas, a mais usual é `length` que retorna o tamanho do array. Quanto aos métodos, há inúmeros. Eis alguns

concat permite incluir novos elementos no array sem precisar saber o índice do último. Exemplo

```
var aqui = [1, 33, 50, "teste"];
var la = aqui.concat(22, "vaiqueda"); // agora
```

Agora o array `aqui` continua com 4 elementos, mas o array `la` passa a ter 6 elementos.

every percorre todos os elementos do array e aplica a cada um deles a função que é passada no método `every`. O IE até a versão 8 não suportava este método. Todos os demais browsers o suportam.

filter Aplica um filtro a todos os elementos do array e retorna um novo array só com os elementos que foram filtrados.

forEach Percorre o array e aplica a função especificada a cada elemento. O resultado dessa aplicação constitue um novo array que se constitui no resultado da aplicação de `forEach`.

indexOf Retorna o índice de um elemento do array. Caso o elemento não exista, retorna -1.

lastIndexOf Igual ao anterior, mas se houver mais de um elemento igual no array, retorna o índice do último e não o do primeiro.

join Transforma todos os elementos de um array em uma string.

push acrescenta os argumentos dados no final do array e retorna o número total de elementos

```
var oba = [1, 2, "aaa", "bbb"];
var qtd = oba.push(4, 5, "ccc");
// agora qtd = 7 e oba = [1, 2, "aaa", "bbb", 4, 5, "ccc"]
```

reverse Inverte a ordem dos elementos do array. Não cria um novo e sim altera o array existente.

shift Remove o primeiro elemento do array e retorna o valor que foi removido

```
var oba = [1, 2, "aaa"];
var prima = oba.shift();
// prima agora é 1, enquanto oba ficou [2, "aaa"]
```

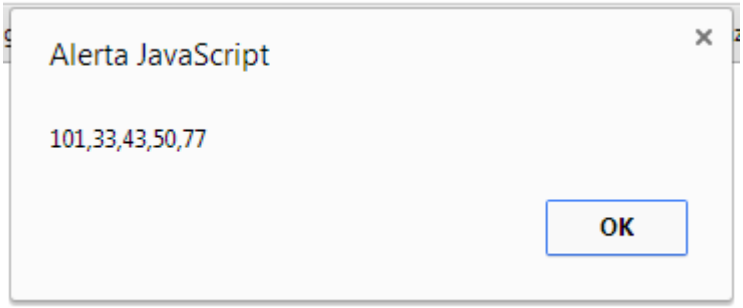
slice retorna um sub-array do array original. Recebe os indices `de` e `até`. Índices negativos começam a contagem do final.

some Percorre todos os elementos do array até que um elemento do array satisfaça as condições impostas no corpo da função fornecida. Ocorrendo isto `some` retorna `true`. Se nenhum elemento do array satisfizer as condições, `some` retorna `false`.

sort Ordena os elementos de um array. Ainda não está plenamente implementada. Veja um exemplo

```
var aqui = [101, 33, 50, 43, 77];
var la = aqui.sort();
alert(la);
```

Deu como resultado



Vale aqui uma explicação. A ordenação sempre é em ordem alfabética. Se os elementos forem números (é o caso), então antes da ordenação eles são transformados em strings. Se for desejada uma ordenação numérica é necssário passar o método uma função que faça isso. É fácil fazer, veja

```
var aqui = [101, 33, 50, 43, 77];
var la = aqui.sort(function(a,b){
  return a-b;
  })
alert(la);
```

Agora a ordenação será a esperada: 33, 43, 50, 77, 101.

6 Funções

A finalidade de uma função é executar uma ação. É usado quando um processamento específico deve ficar confinado a um local ou sobretudo quando a mesma ação deve ser realizada em diversos pontos do programa. Neste casop define-se a função em um local único e depois ela é chamada quando necessária. O bloco de código que constitui a função é chamado corpo da função, e a sua referência a ela é através de um nome. O nome agora pode ser chamado em diversos lugares do código sem precisar repetir o corpo.

Em Javascript há 3 maneiras de definir uma função, vistas abaixo. Em todos os casos a função definida tem o nome de `poderosa`.

usando function Usa-se a declaração `function`.

```
function poderosa() {
// comandos da função
};
```

Este tipo de função recebe o nome de função estática. Uma vez criada ela não executa nada, tendo que ser chamada. Para chamar uma função usa-se o operador `()`. Se na definição da função forem definidos parâmetros (nomes dentro do parênteses da definição, separados por vírgulas) as mesmas informações deverão ser passadas na chamada. A função pode retornar um resultado, utilizando o comando `return`.

usando um construtor `var poderosa = new function(["arg1", "arg2", ...], "comandos da função");`

O construtor `Function()` é nativo de Javascript e serve para criar um objeto função. Note-se que o construtor não define explicitamente um nome para a função. A função é armazenada numa variável com o nome dado. Funções declaradas deste jeito são chamadas anônimas e dinâmicas.

síntaxe literal
```
var poderosa = function() {
// comandos da função
};
```

Neste caso, as funções são conhecidas como funções expressões e também como funções anônimas.

Em qualquer dos casos, se o retorno é composto por mais de uma informação, pode-se devolver um objeto array. Veja o exemplo

```
<head> ... <script type="text/javascript">
function calcularetangulo(a,b){
   var area=a*b;
   var perimetro=(a+b)*2;
   return[area,perimetro]; }
</script> </head> <body>
...
<button type="button" onclick="var resultados = calcularetangulo(5,10);
alert('Area: '+ resultados[0]); alert('Perimetro: '+resultados[1])">
Executar função </button>
```

Se o resultado retornado é muito grande, talvez seja melhor retornar um objeto ou um array associativo.

6.1 Escopo

Variáveis definidas com a palavra `var` dentro de uma função são locais a esta função e não podem ser referenciadas fora do corpo da função. Se isto acontecer, haverá um erro de variável indefinida. Já variáveis definidas dentro de uma função, sem a palavra `var` são globais e podem ser referidas no corpo da função e também fora dele sem que ocorra erro.

7 Strings

Antes usou-se um string (cadeia) como um tipo de dado. Agora vai-se estudar o string como um objeto examinando suas propriedades e métodos. A definição de string como um tipo de dados é um conjunto de zero ou mais caracteres envolvidos por aspas duplas ou por aspas simples. Já o string como objeto é um receptor de textos. Suas propriedades e métodos permitem manipular os textos neles armazenados. Compatível com estas definições, existem duas maneiras de criar strings. A primeira é atribuir uma string a uma variável

```
var frase = "a galinha pintadinha";
```

A segunda, usa o construtor `String` antecedido da palavra `new`

```
var frase = new String("a galinha pintadinha");
```

As propriedades do objeto string são `length`, `constructor` e `prototype`. A propriedade `length` contém o comprimento da string. Os métodos estão a seguir descritos

charAt(índice) Retorna o caractere da string que ocupa a posição definida pelo índice.

charCodeAt(índice) Retorna a codificação Unicode do caractere da string que ocupa a posição definida pelo índice.

concat(string$_1$, string$_2$, ...,$_n$) concatena duas ou mais strings formando uma única.

fromCharCode(uni$_1$, uni$_2$, ... $_n$) retorar o(s) caractere(s) cujo código Unicode foi fornecido.

indexOf(string) retorna o índice da primeira ocorrência no objeto. Veja o exemplo

```
<head> <script type="text/javascript">
 a = new String("pernas pra que te quero");
 x = a.indexOf('pra');  // x vale 7
 alert(x);
</script> </head>  <body>   </body>
```

lasIndexOf(string) retorna o índice da última ocorrência do objeto.

localeCompare(string) retorna um número negativo se o objeto vem depois que a string (em ordem alfabética). Retorna 0 se são iguais e um positivo se o objeto vem antes. Veja o exemplo

```
<head> <script type="text/javascript">
 a = new String("pernas pra que te quero");
 alert(a.localeCompare("aaa"));  // deu 1
 alert(a.localeCompare("pernas pra que te quero")); // deu 0
 alert(a.localeCompare("pernas")); // deu 1
 alert(a.localeCompare("xxx")); // deu -1
</script> </head> <body>   </body>
```

match(expreg) Retorna as ocorrências da string definida pelo parâmetro *expreg*, que é uma expressão regular. O retorno é um array contendo as ocorrências encontradas ou `null`. Expressões regulares serão estudadas mais a frente.

replace(string$_1$|expreg,string$_2$|função) Procura a string$_1$ ou a expreg no objeto e achando-a a substitui por string$_2$ ou pelo resultado da função declarada. Veja no exemplo

```
<head> <script type="text/javascript">
 a = new String("pernas pra que te quero");
 b = a.replace("pra","para");
 alert(a); // pernas pra que te quero
 alert(b); // pernas para que te quero
 </script> </head> <body>   </body>
```

search(expreg) retorna a posição da primeira ocorrência de expreg.

slice(inicio,fim) cria uma nova string a partir da string atual, começando em início e terminando uma posição antes de fim. Fim é opcional e se omitido vai até o final. Se usados números negativos, a contagem começa no final.

split(separador,limite) cria um array de substrings extraídas da string. Ambos parâmetros são opcionais. O primeiro indica qual o caracter que fará a separação e o segundo especifica a quantidade de itens no array.

toLowerCase() retorna os caracteres da string em minúsculo.

toUpperCase() retorna os caracteres da string em maiúsculo.

8 Objeto Math

Este objeto tem propriedades e métodos estáticos. Não tem a propriedade constructor, logo não é possível usá-lo com o operador `new`.

Suas propriedades armazenam constantes matemática, como em

propriedade	expressão	valor
Math.E	$e=$	2.7182818284590452354
Math.LN10	$log_e 10=$	2.302585092994046
Math.LN2	$log_e 10=$	0.693147180559945
Math.LOG10E	$log_{10} e=$	0.4342944819032518
Math.LOG2E	$log_2 e=$	1.442695040888963
Math.PI	$\pi=$	3.141592653589793
Math.SQRT1_2	$(\sqrt{2})/2=$	0.7071067811865476
Math.SQRT2	$\sqrt{2}=$	1.414213562373095

Os métodos do objeto Math são

abs(x) retorna o valor absoluto do número passado ao método. Veja o exemplo:

```
<head> <script type="text/javascript">
 alert(Math.abs(-44)); // dá 44
 </script> </head> <body>   </body>
```

max(x_1, x_2, **...** , x_n) retorna o maior valor dos passados ao método.

min(x_1, x_2, **...** , x_n) retorna o menor valor dos passados ao método.

round(x) arredonda x para o inteiro mais próximo.

floor(x) devolve o chão de x.

ceil(x) devolve o teto de x.

pow(x, y) devolve x^y.

sqrt(x) devolve $\sqrt{x}$.

random() retorna um pseudorandômico entre 0 e 1, inclusive.

exp(x) retorna e^x.

log(x) retorna $log_e x$.

sin(x) retorna o seno de x que deve estar expresso em radianos.

cos(x) retorna o coseno de x que deve estar expresso em radianos.

asin(x) retorna o arco seno em radianos de x.

acos(x) retorna o arco coseno em radianos de x.

atan(x) retorna o arco tangente em radianos e compreendido entre $-\pi/2$ e $\pi/2$.

atan2(x) retorna o arco tangente em radianos e compreendido entre $-\pi$ e π. Este método é usado para converter coordenadas polares em cartesianas.

9 Objeto Number

Permite tratar um número como objeto. Suas propriedades: MAX_VALUE, MIN_VALUE, POSITIVE_INFINITY, NEGATIVE_INFINITY. Seus métodos: toString([base]), toFixed(x), toPrecision(x), toExponential(x).

10 Objeto RegExp

Uma expressão regular é uma maneira otimizada de expressar um padrão de texto. É um conjunto de símbolos que resume um padrão que vai ser buscado em um texto. O objeto RegExp é um construtor de expressões regulares. Possui duas síntaxes, a saber

```
var texto = new RegExp(padrão ],modificador])
```

ou então a síntaxe literal

```
var texto = /padrão/modificador
```

Nos dois casos o *padrão* é a expressão que define a busca. Na síntaxe literal ele começa e termina por uma barra. O *modificador* é opcional e se presente dá opções na busca, que podem ser

i a busca não faz distinção entre maiúsculas e minúsculas.

g busca todas as ocorrências e não só a primeira.

Uma expressão regular (ER) é um método formal de se especificar um padrão de texto. É um conjunto de símbolos e caracteres que quando analisados determinam uma regra. A este conjunto chamamos padrão. Depois, verifica-se um texto maior para procurar pedaços dele que obedeçam à regra criada no padrão. Ao texto maior, chama-se texto. O objetivo é localizar (contar, substituir,...) ocorrências do padrão dentro do texto. Qualquer editor de textos faz isso, por exemplo quando você tem uma longa carta e manda o editor trocar 'Curitiba' por 'São Paulo'. Ao usar expressões regulares, tal processo ganha inteligência e abrangência. Por exemplo, com o padrão = ((su|hi)per)?mercado, você vai recuperar todas as ocorrências das palavras "mercado", "supermercado"e "hipermercado".

Histórico As expressões regulares começaram na teoria da linguística, quando o matemático Stephen Kleene descreveu o que ele chamou de *regular sets* e sugeriu alguns dos símbolos que usamos até hoje. Em 1968, criou-se um editor de textos (*ed*) que passou a fazer buscas usando expressões regulares. Em 1986, apareceu um pacote C livre, chamado regex, que permitia incluir o manuseio de ER dentro de qualquer programa. Finalmente o WORD, a partir da versão 2000 aceita usar ER (devidamente modificadas pela Microsoft, já que ela sempre consegue piorar algo no que mete a mão) em buscas e trocas.

Metacaracteres Para construir o padrão, usamos alguns metacaracteres que controlam a busca do padrão. Eis os principais

caracter	mnemônico	função
.	ponto	um caractere qualquer
[ ]	lista	conjunto de caracteres aceitos
[^]	lista negada	conjunto de caracteres rejeitados
?	opcional	0 ou 1 ocorrência
*	qualquer	de 0 até qualquer ocorrência
+	pelo menos 1	1 ou mais ocorrências
{n,m}	de...até	de n até m
^	início	início da linha
$	fim	final da linha
\b	borda	início ou final da palavra
\c	escape	torna literal o caractere c
|	ou	ou um ou outro
( )	grupo	delimita um grupo
\1 a \9	retrovisor	texto casado nos grupos 1 a 9

. É o maior coringa. O ponto casa com qualquer outro caractere, inclusive com o caractere ponto. Veja no exemplo:

```
n.o          não, nao, neo, noo,...
ca.a         casa, cata, capa,...
e.tendido    estendido, entendido, extendido,...
```

lista A lista é um conjunto de caracteres escritos entre chaves. Ela descreve os caracteres que farão parte do padrão (usando OU). Veja

```
n[aã]o       não, nao - mas deixa de fora o noo
17[:.h ]20   17:20, 17h20, 17.20, 17 20
<[BIP]>      <B>, <I>, <P>
```

Note que a especificação `[0123456789]` pode e deve ser trocada por `[0-9]`. Usa-se também `[a-z]`.

Já a lista negada tem a mesma lógica, mas com o conectivo lógico $\sim$ na frente. Aqui a negação é escrita com `^`. Então, `[^a-z]` é qualquer caractere que não seja uma letra minúscula.

Opcional Usa o símbolo `?` que significa 0 ou uma vez o que vem antes. Então, os exemplos:

```
bate[ri]?     bater, batei, bate, mas não bateu
</?[BIP]>     <B>, </B>, <I>, </I>, <P> ou </P>
```

Tanto faz O tanto faz é indicado pelo `*` e ele indica zero vezes, uma vez, ou um monte de vezes o que vem antes.

```
5*0            0,50,550,55555555550,...
va*t           vt, vat, vaaaaaat,...
b[hj]*k        bk, bhjhhhk, bhjhjhjjjjk, ...
```

Note que `.*` significa qualquer coisa em qualquer quantidade. Então, a pesquisa `prazo.*semana` vai procurar a linha de texto em que alguém prometeu terminar no prazo máximo de 3 semanas.

Pelo menos um Identificado pelo símbolo `+` funciona igual ao `*`, mas tem uma diferença: o que vem antes do `+` tem que ocorrer pelo menos 1 vez. Compare:

```
5+0            50,550,55555555550,...
va+t           vat, vaaaaaat,...
b[hj]+k        bhjhhhk, bhjhjhjjjjk, ...
```

Controle {n,m} É uma quantificação mais controlada, onde `{n,m}` singifica um quantificador de no mínimo n e no máximo m. Por exemplo `3{2,4}` equivale a 33, 333, 3333 e só. Acompanhe os exemplos

```
{2,3}     de 2 a 3 vezes
{4,}      pelo menos 4 vezes
{0,3}     até 3 vezes
{4}       exatamente 4 vezes
{1}       1 vez
{0,1}     0 ou 1 (igual ao ?)
{0,}      zero ou mais (igual ao *)
{1,}      um ou mais (igual ao +)
```

Âncora início Usa o caractere `^` (fora de `[]` – dentro é a negação da lista). Indica que o casamente deve ocorrer NO INÍCIO DA LINHA. Exemplo: `^[0-9]` significa case um número no começo da linha. Outro exemplo `^[^a-z]` significa case qualquer linha que não comece com uma letra minúscula. Mais um: `[0-9]^`: Como não tem sentido haver algo antes do início da linha (se tiver algo antes do início da linha, então não é o começo da linha...) Neste caso o `^` é exatamente o caractere circunflexo. Para terminar: `^^` acha todas as linhas que começam com `^`.

Âncora fim O símbolo é `$` e sinaliza o final de linha. Exemplo: `[0-9]$` casa com linhas que terminam por um número. Outro: `^$` significa uma linha vazia. Mais: `^.{10,50}$` casa com linhas que tenham entre 10 e 50 caracteres.

Borda de palavra Marca os limites de uma palavra. Seu símbolo é `\b`. Palavra engloba apenas `[A-Za-z0-9_]`, caracteres fora daqui não fazem parte de palavras. Veja os exemplos

```
dia       dia, diafr, radial, melodia, meio-dia!
\bdia     dia, diafr, meio-dia!
dia\b     dia, melodia, meio-dia!
\bdia\b   dia, meio-dia!
```

Escape Escapa-se um caractere (escrevendo uma barra invertida antes do caractere) quando o mesmo deve ser procurado literalmente. Por exemplo, suponha que você precisa procurar literalmente `viva*`. Se escrever `viva*` isso vai casar com `viv`, `viva`, `vivaaaaa`, cadê o asterisco ? Tudo errado. Um jeito aceitável seria `viva[*]`, mas fica esquisito ter uma lista de um único caractere. A melhor solução é escapar o `*`. Vale para qualquer metacaractere, inclusive a `\` que neste caso deve ser escrito como `\\`.

Ou Partes alternativas do padrão. Por exemplo `aqui|ali` casa com o padrão `aqui` ou com o padrão `ali`. Lembre que `[ab]` casa com `a` ou com `b`, mas dentro de uma lista o OU funciona apenas para 1 caractere. Se o padrão tem mais de 1 caractere, você precisa usar o `|`. Compare `[gtr]ato` equivale a `gato|tato|rato`.

Grupo Escrito entre parênteses, serve para juntar coisas. Por exemplo: `(oi)+` casa com `oi`, `oioi`, `oioioioi` e assim por diante.

```
(\.[0-9]){3}       .0.5.4, .2.1.1, .3.3.3, ...
(www\.)?aa.com     www.aa.com, aa.com
(in|con)?certo     incerto, concerto, certo
```

Retrovisor O retrovisor usa o caracter `\`. Logo a barra invertida escapeia (que palavra feia, e a propósito, inexistente) qualquer caractere EXCETO se for um número de 1 a 9. Então `\1` é o último pedaço do texto que casou. `\2` é o penúltimo, `\3` é o ante-penúltimo e assim por diante até o nono último. Veja o exemplo: `(binho)-\1` busca o casamento de `binho-binho`. Outro: `([A-Za-z]+)-?\1` casa com `quero-quero`, `bibi`, `dudu`, `bombom`... . Como lembrete final, o retrovisor referencia o texto casado e não a ER do grupo.

Metacaracter tipo barra-letra Além daquela tabela da linguagem C (`\a` = bipe; `\n` = newline, `\b` = backspace, ...) criaram-se novos conjunto barra alguma coisa, a saber

`\d`	dígito
`\D`	não dígito
`\w`	palavra
`\W`	não palavra
`\s`	espaço
`\S`	não espaço

Tem uma sequência enorme de barra-alguma coisa. Cada aplicativo|linguagem aplica os seus. Confirme antes de usar!

Exemplos

Seja um texto
`12345678901234567890123456789012345678901234567`
`HDHEIJCEIGCDAACBFCEBBFGBEBJGFHEEIJGIFBEFJGFEACE`
Com um padrão igual a `HE+IJ` Suas ocorrências serão em 3 30.

`12345678901234567890123456789012345678901234567`
`EHGBFJIGDDJHJGEGBCIGBFFFGCIBIEGDJFCACCCIFGBAHHJ`
Com um padrão igual a `CIF?GB` Suas ocorrências serão em 18,39.

`12345678901234567890123456789012345678901234567`
`AJFEEHICJFCIAEIJDIIICBCBIFBGJGIJBCCCBAICFBGBDDF`
Com um padrão igual a `B[C|C|G]+B` Suas ocorrências serão em 22 33 42.

10.1 Expressões Regulares em Javascript

Para examinar alguns exemplos, precisamos uma passada rápida por alguns métodos deste objeto. São eles:

test(str) Responde `true` se um determinado padrão existe em str. Exemplo:

```
<head> <script type="text/javascript">
var pasargada = "LA SOU AMIGO DO REI";
var padrao = /[A-Z]O/g;
alert(padrao.test(pasargada));  // responde true
 </script> </head> <body>   </body>
```

exec(str) Retorna o texto casado na busca

replace(expreg,novastring) substitui os casamentos pela novastring. Exemplo:

```
<head> <script type="text/javascript">
<head> <script type="text/javascript">
var pasargada = "LA SOU AMIGO DO REI";
var padrao = /[A-Z]O/g;
alert(pasargada.replace(padrao,"**"));  // dá LA **U AMI** ** REI
 </script> </head> <body>   </body>
```

search(expreg) retorna a prosição da primeira ocorrência de expreg.

```
<head> <script type="text/javascript">
var pasargada = "LA SOU AMIGO DO REI";
alert(pasargada.search(/AM/)); // da 7
alert(pasargada.search(/[0-9]/)); // da -1
 </script> </head> <body>   </body>
```

match(expreg) Retorna as ocorrências de expreg dentro do texto. Exemplo:

```
<head> <script type="text/javascript">
var pasargada = "LA SOU AMIGO DO REI";
alert(pasargada.match(/[A-Z]O/g)); // da SO, GO, DO
 </script> </head> <body>   </body>
```

11 Objeto Date

O objeto `Date` serve para manusear carimbos temporais: a saber, data e hora. Existem 4 construtores para o objeto Date, são eles

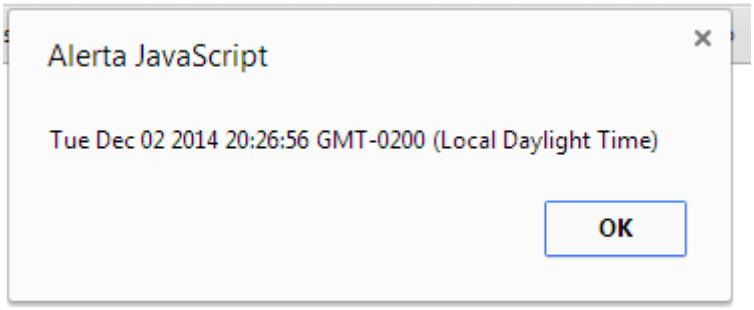
```
var d1 = new Date();
var d2 = new Date(milissegundos);
var d3 = new Date(data\_string);
var d4 = new Date(ano, mes, dia, hora, minuto, segundo, milissegundo);
```

Todos eles manuseiam a data e hora do sistema operacional onde rodam, razão pela qual esta informação não pode ser 100% confiável. Todas as contagens de tempo usam a data referencial de 1^o de janeiro de 1970, às 00h00min00seg. Esta data é conhecida como Unix Time.

Se chamarmos o construtor Date sem passar nenhum parâmetro a ele (formato `d1` acima), está-se solicitando a data hora local. Veja o exemplo

```
<head> <script type="text/javascript">
var now = new Date();
alert(now);
 </script> </head> <body>   </body>
```

dando o seguinte resultado:



Note que os dias e meses sempre vem em inglês, independentemente do idioma de instalação do sistema operacional.

Ao usar o formato `d2` acima, o javascript calcula uma data contando a quantidade de milissegundos para a frente de 01/01/1970 se a quantidade for positiva e para trás da mesma data se a quantidade for negativa.

Se a formato `d3` for o usado, deve-se escolher uma de 5 síntaxes

```
MM-dd-aaaa hh:min:seg
aaaa/MM/dd hh:min:seg
MM/dd/aaaa hh:min:seg
MMM dd, aaaa hh:min:seg
MMMMMMMMM dd, aaaa hh:min:seg
```

Se o grupo `hh:min:seg` for omitido, assume-se `00:00:00`. `MMM` é a abreviatura em inglês do mês (Jan, Fev, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec) e `MMMMMMM` é o nome completo do mês em inglês. Note-se que a contagem dos meses, segue a contagem padrão de javascript. Assim, o mês de janeiro é o mês zero, Fevereiro é 1 e dezembro é 11.

11.1 Métodos do objeto Date

getDate() retorna o dia do mês de um objeto data.

```
<head> <script type="text/javascript">
var now = new Date();
alert(now.getDate());  hoje é 2/dez/14 e a resposta foi 2
 </script> </head> <body>   </body>
```

getDay() retorna o dia da semana (0-6) de um objeto data.

getFullYear() retorna o ano com 4 dígitos.

getHours() retorna a hora (0-23) do objeto data.

getMilliseconds() retorna a quantidade de milissegundos (0-999) do objeto data.

getMinutes() retorna os minutos (0-59) do objeto data.

getMonth() retorna o Mês (0-11) do objeto data. Para obter o nome do mês, possivelmente em português, construa um array com os nomes deles todos e use como índice a resposta a esta chamada.

getSeconds() retorna a quantidade de segundos (0-59) do objeto data.

getTime() retorna a quantidade de milissegundos desde 01/01/1970.

getTimezoneoffset() retorna a diferença em minutos entre a hora local e a hora do meridiano de Greenwich (GMT)

toDateString() Converte a data (sem hora) para string.

toTimeString() Converte a hora (sem data) para string.

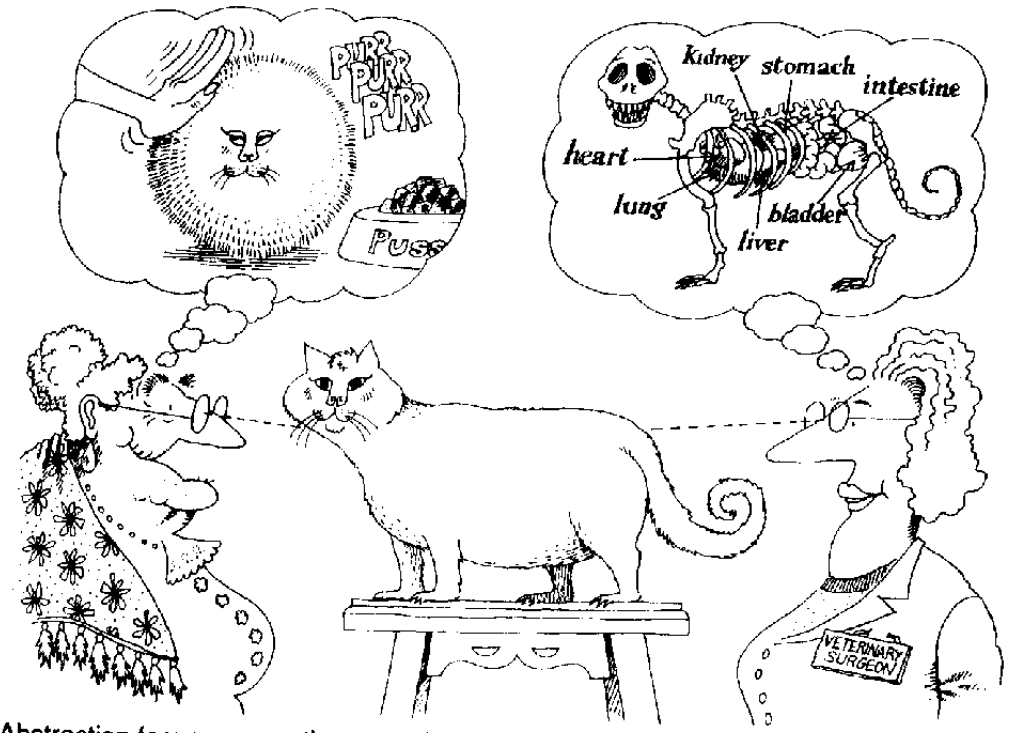
12 Objeto BOM=browser object model

Este objeto descreve o navegador para um programa javascript. Ele potencializa uma dificuldade sempre presente no ambiente javascript, que é como cada navegador implementa cada coisa. Vale sempre a regra: teste tudo em todos os ambientes. O BOM deve ser – sempre que possível – substituído pelo DOM = Document Object Model. Note-se também que BOM não está descrito no ECMAScript. Muitas de suas funcionalidades têm seus equivalentes no DOM (este sim, descrito no ECMAScript).

13 DOM

Subdivide-se em 3 subconjuntos: Core, HTML e XML.

Vê-se aqui como uma madame e uma veterinária veem ambas o mesmo gato. Imagine que o gato é o nosso documento HTML. Alguém pode vê-lo como uma série de ordens de formatação e conteúdo para o computador cliente. Outro pode vê-lo como uma estrutura abstrata de um documento. É esse exercício que se fará agora.



Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

Um exercício: como seriam as visões do mesmo gato para outros atores:

- um rato

- um funcionário da carrocinha
- um funcionário de uma pet-house interessado em vender uma ração premium (bem cara)
- um cachorro briguento
- um juiz de uma competição de raças felinas
- a Agatha (namorada do Gaturro)
- o carteiro do Garfield
- o Jerry
- o Guarda Belo

13.1 DOM-HTML

O DOM HTML é uma representação da estrutura do documento HTML. É uma estrutura em árvore, onde a raiz é o nodo HTML, seus filhos são HEAD e BODY, os filhos de BODY são H1, P, UL, ... O objeto document representa um documento aberto no browser. Toda vez que este abre um arquivo HTML é criado um objeto deste tipo sob o Javascript. Este pode ser acessado usando a síntaxe `window.document`. O objeto documento permite que se acesse, via Javascript, todos os elementos HTML da página.

13.2 DOM-Core

O DOM-Core é uma API que permite ao programa Javascript acessar e manipular conteúdos HTML e XML. Aqui o documento é descrito de maneira mais abstrata do que no caso do HTML. Os nodos são document, element, attribute, comment, text, e assim por diante.

13.3 DOM-Estilos

Esta é uma interface que representa qualquer tipo de folha de estilos.

14 Eventos

São os eventos que tornam Javascript útil. Para realizar a ação nele prevista, um script depende da ocorrência de um evento. Aqui não se pode esquecer da guerra de browsers. Existe a implementação W3C, a da Microsoft e por razões históricas a implementação da Netscape. Esta última é a mais bem suportada – também por razões históricas. Eis algumas definições importantes

evento algo que ocorre sempre que há uma interação do browser (na verdade do usuário do) com um elemento da marcação HTML. Os nomes dos eventos estão em inglês e são descritivos do evento. Um evento é um objeto dentro de Javascript. Por exemplo *mousemove, mousedown, mouseup, click, load, unload*, etc. As propriedades do objeto evento são denominadas acrescentando *on* ao nome do evento. Então, *onmousemove, onload*, etc.

manipulador Ou em inglês, *handler* é uma função que é executada quando o evento ocorre.

disparador é o elemento HTML ao qual está associado o manipulador.

manipulador padrão é uma função padrão do browser que é executada sempre que ocorrem eventos. Quando o programador define um manipulador específico este é executado sempre antes do manipulador padrão. Então, se o usuário quiser, pode cancelar a manipulação padrão, bastando neste caso, retornar falso no seu manipulador.

14.1 Associando um manipulador a um evento

Existem duas maneiras de fazer isto, veja-se: Na primeira, dentro do próprio elemento descreve-se o que fazer:

```
<html> ... <body>
<button id="botao 1" onclick = "alert('Fui apertado')">
Clique aqui </button> </body> </html>
```

A segunda maneira (melhor) é

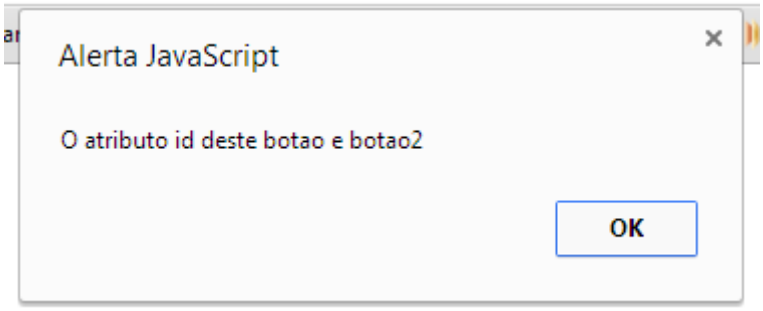
```
<html> <head> <script type="text/javascript">
function mostrando() {
  alert("fui apertado 1"); } </script> </head> <body>
  <button id="botao 1" onclick = "mostrando()">
  Clique aqui 1</button> </body> </html>
```

14.2 Manipulador no DOM 2

Esta abordagem destina-se a deixar mais estanque a definição do que fazer ao ser apertado um botão. Note que a definição do botão no HTML não faz referência a nenhum evento. Isto fica exclusivamente para a camada Javascript. Para o munido IE, esta implementação só está disponível a partir do IE9.

```
<html> <head> <script type="text/javascript">
window.onload = function(){
   var botao = document.getElementById("botao2");
   botao.addEventListener("click", function() {
   alert("O atributo id deste botao e " + this.id); }, false); } </script> </head>
<body>
  <button id="botao2" >
  Clique aqui2</button> </body> </html>
```

deu como resposta



14.3 Eventos do mouse

São: click, dblclick, mousedown, mouseup, mouseover, mouseout, mousemove. Suas propriedades são clientX, clientY, screenX, screenY, relatedTarget, shiftKey, ctrlKey, altKey, metaKey, button. Veja um exemplo

```
<html> <head> <script type="text/javascript">
var EventUtil = {
  getEvent: function(event) {
  return event ? event : window.event;
  }
}
window.onload = function() {
    var event = EventUtil.getEvent(event)
    var paragrafos = document.getElementsByTagName("p");

    paragrafos[0].onclick = function(event) {
        alert("Evento: " + EventUtil.getEvent(event).type)
    };
    paragrafos[1].onmousedown = function(event) {
        alert("Evento: " + EventUtil.getEvent(event).type)
    };
    paragrafos[2].onmouseup = function(event) {
        alert("Evento: " + EventUtil.getEvent(event).type)
    };
    paragrafos[3].onmouseover = function(event) {
        alert("Evento: " + EventUtil.getEvent(event).type)
    };
    paragrafos[4].onmouseout = function(event) {
        alert("Evento: " + EventUtil.getEvent(event).type)
    };
    paragrafos[5].ondblclick = function(event) {
        alert("Evento: " + EventUtil.getEvent(event).type)
    };   }
</script> </head>
<body>
<p>Aqui click</p>
<p>Aqui mousedown</p>
```

```
<p>Aqui mouseup</p>
<p>Aqui mouseover</p>
<p>Aqui mouseout</p>
<p>Aqui duplo clique</p>
</body> </html>
```

14.4 Eventos de teclado

São eles: keydown, keypress, keyup. As propriedades são shiftKey, ctrlKey, altKey, metaKey, keyCode, charCode.

14.5 Eventos HTML

São: load, unload, abort, error, resize, scroll, focus, change, select, submit, reset.

14.6 Eventos de mutação

Ocorrem quando há uma mudança no DOM. Ainda é mal suportado.

14.7 Eventos Proprietários

Implementados por fabricantes e desenvolvedores de software destinados a funcionar exclusivamente com seus dispositivos. Herança da guerra dos navegadores. Obviamente estes eventos não são *crossbrowser*. Em particular aqui há uma peleja entre os padrões WEB e a Microsoft (através de seu IE).

15 Formulários

Um formulário é uma peça de composição de dados presentes em boa parte dos web sites para captura de dados, máquinas de busca, etc. Formulários por sí só já são interativos, prontos para serem manipulados sem a necessidade de scripts. Entretanto, uma regra de ouro em informática é a presunção do *digitador macaco* [2]. Por esta suposição, deve-se esperar um dado completamente oposto ao que se desejaria. Exemplos:

CPF deve-se aceitar apenas 11 dígitos, eventualmente um ou outro sinal separador.

telefone apenas dígitos, eventualmente com código DDD, e com tamanho pré-definido (quanto ?). Um campo obrigatório ?

estado uma escolha em uma lista

cidade um nome ou uma lista ?

nome de alguém um campo obrigatório ?

Então, o Javascript pode ser acionado para validar atributos digitados. Deve-se ressaltar entretanto que não se pode confiar exclusivamente no Javascript para fazer isto (o usuário do computador cliente sempre pode desativar o Javascript) devendo-se refazer as verificações do lado servidor. A finalidade da verificação no computador cliente é apenas apontar erros involuntários cometidos pelo digitador, no instante mesmo em que eles ocorrem. Vale lembrar outra regra de ouro da informática *Quanto menos tempo decorre entre o cometimento de um erro e o seu apontamento, mais rápida (e barata) é a sua correção.* O elemento HTML destinado a marcar um formulário é a tag `<form>` com o seu fechamento em `</form>`. Os atributos de `form` são

`action` Uma URL onde está o programa que vai tratar os dados enviados pelo formulário

`method` métodos `get` e `post`.

`name` Nome do formulário. Em desuso, usar o atributo `id`.

Outros enctype, accept-charset, accept, id, class, lang, style, title, target, onsubmit, onreset, onclick, ondblclick, onmousedown, onmouseover, onmouseout, onkeypress, onkeydown, onkeyup.

15.1 Controles

Os elementos HTML destinados a marcar os controles (campos) e seus atributos:

[2] Quando você programar uma entrada de dados, nunca pode assumir que o digitador teclou exatamente o que você espera. Faça melhor: suponha que o seu digitador é um macaco – com QI típico de macaco. Você vai se dar muito melhor na profissão.

15.1.1 input

Esta tag é vazia (não tem terminador) e define uma entrada de dados. O tipo é definido pelo atributo `type`.

15.1.2 type

Define o tipo da entrada. Seus valores:

`text` entrada de texto em uma linha

`password` igual ao anterior, mas sem ecoar os dados

`checkbox` cria uma caixa de seleção (verdadeiro ou falso)

`radio` um botão de rádio

`submit` um botão para o envio do formulário

`reset` um botão que retorna o formulário à condição inicial

`file` cria um campo para a entrada de um arquivo local a ser enviado ao servidor

`hidden` um campo de entrada cujos dados não são visíveis ao usuário

`image` um botão igual ao submit, mas que é visto com uma imagem dentro

`button` um botão que aciona um script para uma determinada ação