JavaScript

referências à linguagem

Livro escrito no Wikibooks em língua portuguesa, livre pensar e aprender.

Esta obra é o resultado da experiência de várias pessoas, que acreditam que a melhor forma do conhecimento é o conhecimento compartilhado. Aqui temos uma pequena referência a estas pessoas:

- Sérgio Eduardo Rodrigues
- Lightningpirit

Este livro está para download livre e gratuito neste endereço:

- http://pt.wikibooks.org/wiki/Javascript

A versão on-line e em contínuo desenvolvimento está nesse endereço:

- http://pt.wikibooks.org/wiki/Javascript

- A versão para este arquivo PDF foi modificada pela última vez em 30 de Julho de 2006, modificações posteriores a esta data no site do Wikibooks não estão inclusas.

- Este arquivo PDF foi feito através do software OpenOffice.org versão 2.0.3.

ÍNDICE:

INTRODUÇÃO

Muito se fala sobre Javascript, porém existe muito pouca documentação em português desta linguagem de programação.

O maior problema é a confusão que se faz entre a linguagem Java e o Javascript com relação a semelhança no nome, na mesma linha de raciocínio fazem a relação de C++ com Java pela semelhança de comandos e sintaxes, embora a implementação de algumas sintaxes e construções sejam parecidas são linguagens distintas, criadas para resolver problemas diferentes, e que por isto possuem capacidades diferentes, o mesmo ocorre com Java e Javascript.

Enquanto a Linguagem Java é fortemente tipada e possui tipos estáticos o Javascript oferece uma reduzida quantidade de tipos, isto é, o necessário para criação de telas dinâmicas e certa lógica as páginas html.

Javascript é pequena, leve, portável (está presente em quase todos os navegadores e em todas as plataformas que estes navegadores rodam). Ela não é uma linguagem para rodar sozinha, precisando de um navegador para tal.

Javascript segue uma linha de quanto menor melhor, ou seja, ela é pequena na sua escrita, e criação de objetos. Os tipos de variáveis são dinâmicos, e possui objetos globais já predefinidos conforme o ambiente que se encontra.

Inicialmente ela foi criada pela Netscape para ser a linguagem padrão no navegador, para produzir certa verificação aos dados. Porém com o passar do tempo, ela foi padronizada pela ECMA(European Computer Manufactures Association) (http://www.ecma-international.org) vide ECMA-262, também reconhecida pela ISO ISO-16262.

Porém esta padronização não avançou sobre os objetos públicos e seus respectivos métodos, o que torna difícil a padronização de programas entre navegadores.

Aqui será abordada a a especificação JavaScript 1.5 que segue a ecma262 edição 3.

TIPOS DE DADOS

O Javascript possui poucos tipos de dados, sendo eles:

- ## Numéricos:

Este tipo de dado armazena valores, tanto valores inteiros como ponto flutuante, por exemplo:

- 1
- 84
- 2e10
- 3.141516
- 0.000001

Os valores numericos podem fazer parte de operações aritimética, como a soma, subtração, multiplicação e divisão.

Valores numéricos também podem fazer parte de operações aritiméticas de bits. Como por exemplo (>>) rotação de bits para direita, (<<) rotação de bits para esquerda, (>>>) rotação de bits a direita sem levar em conta o sinal, (^) OU exclusivo (= XOR), (&) operação E binária (= AND), (|) operação OU binária (= OR), (~) Negação binária (= NOT).

Existem várias funções para manipulação de valores numéricos, como funções trigonométricas, funções de arredondamento e exponenciação, funções de transformação de tipos, etc.

Existem alguns valores numéricos especiais, são eles :

- NaN - Na verdade NaN é a abreviação de (*Not a Number*) = (Não um Número), ele é o resultado geralmente de operações inválidas com números. Como por exemplo, o resultado da operação (0/0), irá resultar no NaN. Ele também é uma constante, que pode ser atribuida a uma variável, como veremos mais adiante.
- Infinity - Representa um valor infinito, podendo ser tanto positivo quanto negativo. Todas as operações com valores infinitos resultarão num valor infinito, exceto divisão e subtração que resultará no NaN.

- ## Lógicos:

Os valores lógicos podem assumir dois valores, *true* (verdadeiro) e *false* (falso).

- ## Strings:

São cadeias de caracteres, o maior número que uma string pode conter depende do navegador em uso.

Valores strings são valores delimitados por apóstrofo(') ou por aspas("), ex: "texto" ou 'texto' uma barra inversa permite a inserção de caracteres especiais, ex:

"\b" - Representa o backspace (caracter ascii 8)
"\t" - Tabulação (caracter ascii 9)
"\r" - Retorno de carro (caracter ascii 13)
"\n" - Nova linha (caracter ascii 10)
"\v" - Tabulação vertical (caracter ascii 11)
"\uNNNN" - Caracter unicode (NNNN representa um valor hexadecimal de 0000 a FFFF)
"\xNN" - Caracter ascii (NN representa um valor hexadecimal de 00 a FF)
"\'" - Apóstrofo
"\"" - Aspas
"\\" - Barra inversa

- Null:

O null é um valor especial, representa um objeto nulo, não deve ser confundido com uma variável não inicializada, pois o valor null existe.

Portanto uma variável com o conteúdo null existe em memória, referenciando este objeto especial.

- Undefined:

O valor undefined significa que a variável não foi instanciada, inicialmente todas as variáveis se encontram neste estado.

CONVERSÃO DE TIPOS

Atribuindo valores

Diferente da maioria das linguagens o javascript define as variáveis dinamicamente, portanto ao atribuir uma variável ele escolhe o tipo conforme o valor passado para a variável, não sendo necessário especificar o mesmo.

```
var numero = 1;
var texto = "Sérgio Eduardo Rodrigues";
var valor = 123.45;
var ativo= true;
var nascimento = new Date(1969,1,4)
```

Nome	Tipo
numero	numerica inteira
texto	string
valor	numerica com ponto flutuante
ativo	booleana
nascimento	objeto date

Convertendo

Uma variável pode ser atribuida para outro tipo, ou utilizando uma função de conversão ou então fazendo operações aritiméticas.

Como por exemplo, quando atribuimos ao **numero** o valor 1, ele se encontra no tipo numérico inteiro, se o dividirmos por 2 ele irá para o tipo numérico ponto flutuante:

```
numero = 1; // inteiro 1
numero = numero / 2; // Vai para flutuante 0.5
numero = " " + numero; // ele é convertido para string, pois está sendo somado à outra string
numero = parseFloat(numero); // Ele irá resultar no número 0.5
numero = parseInt(numero); // Vai para o inteiro 0
```

VARIÁVEIS: DEFINIÇÃO:

As variáveis são representadas por nomes chamados de identificadores, este identificadores tem certa regra para ser montado:

1° Deve iniciar obrigatoriamente por letra ou pelo símbolo "_" ou "$"
2° A partir daí além de letras, "_" e $ pode conter dígitos(0 até 9).

você pode declarar uma variável de duas formas:

1ª Atribuindo diretamente a variável,
exemplo
nome="Sérgio";
2ª Utilizando a palavra reservada "var",
exemplo
var nome = "Sérgio Eduardo rodrigues";

Uma variável ou matriz que não tenha sido inicializada, possui o valor de "undefined", observe que o Javascript é sensível para *case-sensitive*, ou seja, **letras minúsculas e maiúsculas são diferentes**, portanto, undefined e null devem ser escritos sempre em letra minúsculas.

Se uma variável é declarada apenas, com o comando var, o seu conteúdo é "undefined", ou NaN(*Not a Number*), caso esteja num contexto numérico.

Exemplo:

```
var x;
x = x * 2;
```

o resultado será NaN.

ou se for utilizado

```
x = x + "teste"
```

causará um erro de execução, pois x não tem valor definido.

VARIÁVEIS: ESCOPO:

Caso a variável seja declarada fora do corpo de uma função ela será considerada como pública, ou seja poderá ser alcançada por todas as funções, caso ela seja declarada dentro de uma função ela é considerada privada, pois somente pode ser vista pelo código da função.

Exemplo 1(variável pública):

```
var x=10;
function fx(){
  ... será possível utilizar o valor de x ...
}function fy() {
  ... será possível utilizar o valor de x ...
}
```

Exemplo 2 (variável privada):

```
function fx() {
   var x = 5;
   ... será possível utilizar o valor de x ...
}
function fy() {
   ... x terá valor undefined, ou seja não será visto por fy ...
}
```

VARIÁVEIS: CONSTANTES:

São variáveis declaradas com a palavra chave “const”, que não podem sofrer alteração de seu conteúdo e nem de sua declaração no escopo da rotina. Exemplo:

```
const fator = 1.34;
const nome = “Sérgio”
```

se tentar efetuar uma redeclaração ocorrerá um erro de execução, exemplo:

```
const fator = 1.34;
var fator = 22;
```

ou então se este já tiver sido declarado

```
function funcao() {}
const funcao=”teste”;
```

VARIÁVEIS: MATRIZES:

São variáveis que contém várias ocorrências em seu interior. A declaração de uma variável é feita utilizando ou elementos delimitados por colchetes “[]” ou pelo objeto Array(), exemplo:

```
  var frutas=[“laranja”, “banana”, “pera”];
  var nomes=new Array(“Sérgio”, “Eduardo”,”Rodrigues”);
  var valores=[1.34, 10, 50, 13e2];
```

```
pode-se utilizar elementos vazios na declaração de uma matriz, por exemplo :


 var var frutas=["laranja","banana",,"pera",,,"abacaxi"];
```

resultado :

- frutas[0] = "laranja"
- frutas[1] = "banana"
- frutas[2] = undefined
- frutas[3] = "pera"
- frutas[4] = undefined
- frutas[5] = undefined
- frutas[6] = "abacaxi"

lembre sempre que as matrizes iniciam pelo elemento zero(0).

length é um atributo especial que possui a quantidade de elementos da matriz, não é uma função, ou seja se utilizar frutas.length() ele causará erro.

VARIÁVEIS: NUMÉRICAS:

Existem duas categorias de números, os de notação de ponto flutuante e os inteiros. Os primeiros representam os valores fracionados, com valores decimais, podem ser expressos como:

```
01/01/23
1e3 = 1 x 10³ = 1000
-3.28e12 = 3.28 x 1012 = 3280000000000
1e-12 = 1 x 10-12 = 0.000000000001
```

Valores inteiros, podem ser representados em base 10(decimal), 16(hexadecimal) ou 8(octal). Exemplos :

```
012 igual a 10 decimal, inicia com 0 ele assume que o numero é octal 0x12
igual a 18 decimal, inicia com 0x é assumido como numero hexadecimal.
12 representa 12 decimal.
```

VARIÁVEIS: STRINGS:

São seqüência de caracteres delimitados por (“) aspas ou (') apóstrofe. Exemplos:

'Sérgio Eduardo'
"um texto qualquer"
"várias linhas:\nSegunda Linha\tMesma linha com tabulação"

Ao inicializar uma string pode-se utilizar caracteres especiais, este tem uma barra inversa(\) para indicar que seu significado é especial, veja a seguinte tabela

\b	Representa o backspace (caracter ascii 8)
\t	Tabulação (caracter ascii 9)
\r	Retorno de carro (caracter ascii 13)
\n	Nova linha (caracter ascii 10)
\v	Tabulação vertical (caracter ascii 11)
\uNNNN	Caracter unicode (NNNN representa um valor hexadecimal de 0000 a FFFF)
\xNN	Caracter ascii (NN representa um valor hexadecimal de 00 a FF)
\0NNN	Caracter ascii (NN representa um valor octal de 000 a 0377)
\'	Apóstrofe
\"	Aspas
\\	Barra inversa

OPERADORES:

Operadores para efetuar mudança do conteúdo de uma variável:

Operador	Descrição
=	Atribui valor a uma variável
++	Incrementa valor de uma variável, x++ é o mesmo que x=x+1
--	Decrementa valor de uma variável, x-- é o mesmo que x=x-1

Operadores para comparação de valores:

Operador	Descrição
==	Igual
!=	Diferente
===	Estritamente igual(verifica conteúdo e tipo da variável)
!==	Estritamente diferente(verifica conteúdo e tipo da variável)
<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior ou igual a

Operadores aritiméticos:

Operador	Descrição
%	Módulo
+	Soma
-	Subtração
*	Multiplicação
/	Divisão

Operadores lógicos:

Operador	Descrição
&&	Módulo
||	Ou
!	Não

Operadores de bits:

Operador	Descrição
&	Operação E
|	Operação Ou
^	Operação Ou Exclusivo
~	Operação Não
>>	Rotação de bits para direita
<<	Rotação de bits para esquerda
>>>	Rotação de bits para direita sem levar em consideração o sinal

Operadores especiais:

Operador	Descrição
?:	Efetua operação condicionada, exemplo x = a > 1 ? 3 : 4; ou seja se o valor da variável a for maior que 1, será atribuído a x o valor 3 caso contrario 4
,	A vírgula efetua operação da esquerda para a direita sendo que o último elemento é retornado. Ex: x=1, y=2;
delete variavel	Elimina um objeto que esteja sendo referenciada pela variável, se a variavel for uma propriedade de um objeto, limpa esta referência do objeto.
propriedade **in** objeto	Retorna true caso a propriedade esteja contida no objeto
objeto instanceof TipoDoObjeto	Retorna true caso o objeto seja de determinado tipo
typeof(objeto)	Retorna string contendo o tipo do objeto
New TipoDoObjeto(p1, ...)	Cria uma instância do objeto
This	Representa a instância do objeto corrente
void (expressao)	Resolve expressão, porém ignora valor retornado

Short Circuit

Operações lógicas utilizam short circuit da seguinte forma:

```
true || qualquer coisa = true
false && qualquer coisa = false
```

assim é possível por exemplo fazer a seguinte operação:

```
if (a != 0 && 1/a > 0.5) {
  ....
}
```

assim evita erros de divisão por zero.

Outro exemplo da utilização, é para contornar a diferença dos eventos do explorer e do mozilla/firefox.

```
function listener(event) {
  event = event || window.event;
  ...
}
```

Assim se esta função estiver rodando no IE ou num navegador utilizando Gecko irá rodar da mesma forma.

FLUXO DE CONTROLE:

Controlando o que e quando é executado um comando, faz parte de todas as linguagens, e o javascript não é diferente.

Existem vários comandos, que por isto são tratados como palavras reservadas e portanto não devem ser utilizados como identificadores de variáveis ou constantes.

São eles:

- Bloco de comandos
- if/then/else
- while/do..while
- switch/case
- for
- label
- continue
- break
- throw
- Try/Catch/Finally

FLUXO DE CONTROLE: BLOCO DE COMANDOS:

No Javascript, o bloco de comandos é uma estrutura para agrupar outros comandos.

O bloco de comando começa pelo abre chave "{" e finaliza com o fecha chave "}", o ultimo elemento não necessáriamente necessita de finalizar com ponto e virgula ";", mas se terminar não terá problemas, este ultimo ponto e virgula é opcional.

```
 {
   comando;
   comando;
   ...
   comando
 }
ou

 {
   comando;
   comando;
   ...
   comando;
 }
```

FLUXO DE CONTROLE: COMANDO IF..ELSE:

Talvez um dos comandos mais utilizados em todas as linguagens de programação, o '**if**' é um comando utilizado para tomar a decisão de executar o próximo comando baseado numa espressão lógica, se esta expressão for verdadeira o próximo comando é executado, caso contrário ele é ignorado.

Por exemplo, se for necessário dar um alerta ao usuário, conforme a hora, podemos fazer o seguinte:

```
var hora = new Date().getHours();
if (hora < 12)
  alert("bom dia");
if (hora >= 12 && hora < 18)
  alert("boa tarde");
if (hora >= 18)
  alert("boa noite");
```

Note que as três comparações serão feitas, independente da execução, isto é uma perda de tempo, pois se é de dia não pode ser tarde, neste caso anexamos a estrutura do '**if**' o comando '**else**' que executa o comando a seguir caso o resultado da expressão lógica seja **false**, ficando o nosso código assim.

```
var hora = new Date().getHours();
if (hora < 12)
  alert("bom dia");
else if (hora >= 12 && hora < 18)
  alert("boa tarde");
else
  alert("boa noite");
```

Para tornar mais legível podemos escrever da seguinte forma:

```
var hora = new Date().getHours();
if (hora < 12) {
  alert("bom dia");
} else {
  if (hora >= 12 && hora < 18) {
    alert("boa tarde");
  } else {
    alert("boa noite");
  }
}
```

Assim torna mais legível a o comando executado, sem falar que usando blocos de comandos podemos agrupar mais de um comando.

FLUXO DE CONTROLE: COMANDO WHILE:

- Executa comando enquanto condição resultar em verdadeiro:

  ```
  WHILE (CONDIÇÃO)
  COMANDO;
  ```

- Igual ao anterior, porém o comando é executado pelo menos uma vez, mesmo que condição seja falsa.

  ```
  DO
  COMANDO;
  WHILE (CONDIÇÃO);
  ```

FLUXO DE CONTROLE: COMANDO SWITCH:

Se o conteúdo da variável for igual a constante1 ou constante2 será executado o comando1; se for igual a constante 3 será executado o comando 2; caso contrário será executado o comando 3; note que o comando break força o fluxo sair fora do comando switch.

```
switch(variável) {
   case constante1:
   case constante2:
     comando1;
     break;
   case constante3:
     comando2;
     break;
   default:
     comando3;
 }
```

FLUXO DE CONTROLE: COMANDO FOR:

```
for(inicialização; condição; incremento) comando;
```

Efetua uma inicialização em seguida executa o comando enquanto a condição for verdadeira, após a execução do comando executa a expressão de incremento, ex:

```
 for(var i=0; i < 3; i++)
   alert(i);

é equivalente a :

  var i=0;
  while (i < 3) {
    alert(i);
    i=i+1;
  }
```

FLUXO DE CONTROLE: COMANDO LABEL:

Label:

Label permite que a indicação de uma posição que pode ser utilizado com continue e break para salto dentro de um loop.

FLUXO DE CONTROLE: COMANDO CONTINUE:

Continue; e Continue label;

Salta para loop ou para loop que está após label indicado.

FLUXO DE CONTROLE: COMANDO BREAK:

Break; e Break label;

Sai fora do loop corrente ou do loop que esta após label informado.

FLUXO DE CONTROLE: COMANDO THROW:

throw expressão;

Lança exceção.

FLUXO DE CONTROLE: COMANDO TRY:

Captura qualquer erro que comando lance ou captura exceções conforme expressão.

No primeiro caso comando2 será executado caso comando1 lance uma exceção.

No segundo caso o comando3 será executado caso a exceção lançada pelo comando1 seja igual a expressão1, se for igual a expressão2 o comando 4 será executado, se for lançado uma exceção que não seja igual nem a expressão 1 e nem a expressão 2 será executado o comando2;

```
try {
   comando1;
 } catch(e) {
  comando2;
 }

ou

 try {
   comando1;
 } catch(e if e==expressao1) {
   comando3;
 } catch(e if e== expressão2) {
   comando4;
 } catch(e) {
   comando2;
 }
```

Executa comando1 se este lançar uma exceção executa comando 2 em seguida comando3, se comando1 não lançar nenhuma exceção executa comando3;

```
try {
   comando1;
 } catch(e) {
   comando2;
 } finally {
   comando3;
 }
```

FUNÇÕES:

As funções são declaradas pela palavra reservada “function” seguido pelo identificador seguido por parâmetros delimitados por “(“ abre parêntesis e “)” fecha parêntesis, e do corpo dela que é delimitado por “{“ abre chave e “}” fecha chave, exemplo :

```
function quadrado(x) {
  return x*x;
}
```

Funções Predefinidas

Nome	Descrição
atob(base64)	Converte um texto codificado em base64 para binário. Função inversa ao btoa(texto)
btoa(texto)	Converte um texto para base64. Função inversa ao atob(base64).
decodeURI(url)	Função inversa ao encodeURI
decodeURIComponent(url)	Função inversa ao encodeURIComponent
isFinite(valor)	Identifica se o numero é finito.
isNaN(valor)	Identifica se o valor não é um numero
encodeURI(url)	como o escape ele faz substituições no texto para compatibilizar transferencia em links, mas não faz conversão para os caracteres **!*()'**
encodeURIComponent(url)	como o escape ele faz substituições no texto para compatibilizar transferencia em links, mas não faz conversão para os caracteres **! @#$&*()=:/;?+'**
escape(url)	Ajusta url para que possa ser passada em chamadas e links, convertendo os caracteres especiais para formato hexadecimal e espaço para o sinal de +, não faz mudança nos caracteres **@*/+** que ficam inalterados
eval(expressao)	Interpreta expressão de JavaScript, ex: eval(“1+2”), resultado = 3
parseInt(String) ou parseInt(String, base)	Converte a string num valor inteiro, ou converte uma string na base passada para inteiro.
Number(objeto)	Converte a string num valor ponto flutuante
parseFloat(String)	Converte a string num valor ponto flutuante
String(objeto)	Retorna a representação string do objeto
unescape(url)	Função inversa ao escape(url)

OBJETOS:

OBJETOS: DEFINIÇÃO:

Podem ser declarados com propriedades e valores delimitados por chaves "{}" ou através de funções. Exemplo:

```
var pessoa={
  nome:"Sérgio",
  altura:1.72,
  nascimento:new Date(1969,1,4)
};
```

ou

```
 function objetoPessoa() {
   this.nome = "Sérgio";
   this.altura = 1.72;
   this.nascimento = new Date(1969,1,4)
 }
 var pessoa = new objetoPessoa();
```

obs. O parâmetro de mês na criação do objeto Date inicia com 0, ou seja 1 significa fevereiro.

Para acessar qualquer atributo do objeto, basta informar o nome do objeto seguido por seu atributo após um ponto(.), exemplo:

```
 alert(pessoa.nome);
 alert(pessoa.altura);
 alert(pessoa.nascimento);
```

Também pode-se atribuir métodos a objetos, da seguinte forma :

```
function mostrePessoa() {
  alert(this.nome + "\n" + this.altura + "\n" + this.nascimento);
}
var pessoa={
  nome:"Sergio",
  altura:1.72,
  nascimento:new Date(1969,1,4),
  mostre:mostrePessoa
};


pessoa.mostre();
pessoa.nome = "eduardo";
pessoa.altura = 1.78;
pessoa.nascimento = new Date(1975, 6, 25);
pessoa.mostre();
```

OBJETOS: PROPRIEDADES:

Pode-se acrescentar métodos especiais para tratamento das operações de get e de set, exemplo:

```
var conta={
  valor:0;
  set deposito(x) {
    this.valor += x;
  },
  set saque(x) {
    this.valor -= x;
  },
  get provisao() {
    return valor/4;
  }
};


conta.deposito = 100;
conta.saque = 50;
alert("provisão = " + conta.provisao + "\n saldo = " + conta.valor);
```

OBJETOS: OBJETOS PREDEFINIDOS:

Existem uma serie de objetos que já estão definidos para o desenvolvedor, este objetos são largamente utilizados na linguagem, e são fundamentais para interação com os sistemas.

Os principais são:

• Objeto Array:

Definição

O *Objeto Array*(matriz), pode ser tanto criado implicitamente :

```
var m=[1,2,3,4];
```

Como explicitamente :

```
var m=new Array(1,2,3,4);
```

Este objeto manipula uma coleção de outros objetos.

Atributos

Nome	Descrição
length	Quantidade de elementos que tem a matriz

Métodos

Nome	Descrição

concat	Concatena elementos de duas matrizes, ex:
	`var m=[“sergio”, “eduardo”];`
	`var n=m.concat(“rodrigues”);`
	`n terá valor : [“sergio”, “eduardo”, “rodrigues”]`

join	Junta elementos aos da matriz, ex:
	`var m=[“sergio”, “eduardo”, “rodrigues”];`
	`var n=m.join(“+ ”);`
	`n terá valor : “sergio+ eduardo+ rodrigues”`

pop	Remove o ultimo elemento da matriz, retornando o elemento removido, ex:
	`var m=[“sergio”, “eduardo”, “rodrigues];`
	`var n=m.pop();`
	`n terá valor : “rodrigues” e m:[“sergio”, “eduardo”]`

push	Adiciona elemento a matriz, ex:
	`var m=[“sergio”, “eduardo”];`
	`var n=m.push(“rodrigues”);`
	`n terá valor : “rodrigues” e m:[“sergio”, “eduardo”]`

reverse	Reverte ordem dos elementos da matriz, ex:
	`var m=[“sergio”,”eduardo”,”rodrigues”];`
	`var n=m.reverse();`

	n e m terão: [“rodrigues”, “eduardo”, “rodrigues”]
shift	Remove o primeiro elemento da matriz, retornando o elemento removido, ex: var m=[“sergio”,”eduardo”,”rodrigues”]; var n=m.shift(); n terá valor: “sergio” m terá valor: [“eduardo”, “rodrigues”]

slice(inicio, fim) ou slice(inicio)	Retona uma faixa da matriz, ex:
	
	var m=["a","b", "c","d","e","f","g","h"];
	var n=m.slice(0,2);
	n terá valor: [“a”,”b”]

splice(inicio)
 	
splice(inicio, fim)
 	
splice(inicio, fim, item a inserir ...)
	Adiciona ou remove faixas dentro de uma matriz, ex:
	
	var m=["a","b", "c","d","e","f","g","h"];
	var n=m.slice(0,2);
	n terá valor: [“a”,”b”]
	m terá valor: ["c","d","e","f","g","h"]

sort() sort(funcao de comparacao(a, b))	Ordena a matriz, ex: var m=[“sergio”,”eduardo”,”rodrigues”]; var n=m.sort() n terá valor: [“eduardo”,”rodrigues”,”sergio”] No caso de passar a função de comparação, ela deve retornar menor que zero se a < b, 0 se a=b e maior que zero se a > b

unshift(item, ...)	Adiciona um ou mais elementos ao inicio da matriz, ex:
	
	var m=["sergio", "eduardo", "rodrigues"];
	m.unshift("sr");
	m terá valor: [“sr”, "sergio", "eduardo", "rodrigues"]

- # Objeto Date:

  Métodos

Nome	Descrição
Construtor: Date(ano, mês) Date(ano, mês, dia) Date(ano, mês, dia, hora) Date(ano, mês, dia, hora, minuto) Date(ano, mês, dia, hora, minuto, segundos) Date(ano, mês, dia, hora, minuto, segundos, milesegundos)	Todos os parâmetros são opcionais. Lembre-se apenas que o mês janeiro é representado por 0 e não por 1.
Date.UTC(ano, mês) Date.UTC(ano, mês, dia) Date.UTC(ano, mês, dia, hora) Date.UTC(ano, mês, dia, hora, minuto) Date.UTC(ano, mês, dia, hora, minuto, segundos) Date.UTC(ano, mês, dia, hora, minuto, segundos, milesegundos)	Cria tempo relativo ao UTC. Da mesma forma que o contrutor
Date.parse(texto)	Converte a data para um objeto date, o formato é o mesmo utilizado no toString()
Date.now()	Retorna um objeto date representando o momento, o mesmo que new Date(), porém retorna não um objeto, porém um número.
getDate()	Retorna o dia

getDay()	Retorna o dia da semana
getFullYear()	Retorna um inteiro com o valor do ano
getHours()	Retorna hora
getMilliseconds()	Retorna os milisegundos
getMinutes()	Retorna os minutos
getMonth()	Retorna o mês, janeiro = 0
getSeconds()	Retorna os segundos
getTime()	Numero de milisegundos representando a data
getTimezoneOffset()	Retorna a diferença em minutos do horario local e o UTC
getUTCDate()	Retorna o dia UTC
getUTCDay()	Retorna o dia da semana UTC
getUTCFullYear()	Retorna o ano UTC
getUTCHours()	Retorna a hora UTC
getUTCMilliseconds()	Milissegundos UTC
getUTCMinutes()	Retorna os minutos UTC
getUTCMonth()	Retorna o mês UTC
getUTCSeconds()	Retorna os segundos
getYear()	Retorna o ano menos 1900, ex: 2004 retorna 104, pois 2004 = 104 + 1900
objDate.setMinutes(minutos)	Atribui minutos

objDate.setMinutes(minutos, segundo) setMinutes(minutos, segundo, milessegundo)	
objDate.setUTCHours(hora, minuto, segundo, milesegundo)	Atribui hora UTC
setDate(dia)	Atribui dia do mês
setFullYear(ano) setFullYear(ano, mês) setFullYear(ano, mês, dia)	Atribui ano
setHours(hora) setHours(hora, minuto) setHours(hora, minuto, segundo) setHours(hora, minuto, segundo, milesegundo)	Atribui hora
setMilliseconds(milisegundos)	Atribui milisegundos
setMonth(mes) setMonth(mes, dia)	Atribui mês
setSeconds(segundos) setSeconds(segundos, milesegundo)	Atribui segundos
setTime(milisegundos)	Milisegundos da data
setUTCDate(dia)	Atribui dia do mês UTC
setUTCFullYear(ano) setUTCFullYear(ano, mês) setUTCFullYear(ano, mês, dia)	Atribui ano UTC
setUTCMilliseconds(milisegundos)	Atribui milisegundos UTC

setUTCMinutes(minutos) setUTCMinutes(minutos, segundo) setUTCMinutes(minutos, segundo, milisegundo)	Atribui minutos UTC
setUTCMonth(mes) setUTCMonth(mes, dia)	Atribui mês UTC
setUTCSeconds(segundos) setUTCSeconds(segundos, milesegundo)	Atribui segundos UTC
setYear(ano) setYear(ano, mês) setYear(ano, mês, dia)	Atribui ano, numero que menos 1900 do ano atual
toLocaleDateString()	Converte para string apenas a parte da data
toLocaleTimeString()	Converte para string apenas a parte da hora
toUTCString()	Converte para string UTC

- # Objeto Math:

  Atributos

Nome	Descrição
E	valor da constante e (2.718281828459045)
LN10	logarítimo natural de 10 (2.302585092994046)
LN2	logarítimo natural de 2 (0.6931471805599453)
LOG2E	logarítimo de e na base 2 (1.4426950408889634)
LOG10E	logarítimo de e na base 10(0.4342944819032518)
PI	valor do pi (3.141592653589793)
SQRT1_2	raiz quadrada de 1/2 (0.7071067811865476)
SQRT2	raiz quadrada de 2(1.4142135623730951)

Métodos

Nome	Descrição
abs(v)	valor absoluto de v
acos(v)	arcocosseno de v
asin(v)	arcoseno de v
atan(v)	arcotangente de v
atan2(y, x)	arcotangente de y/x
ceil(v)	próximo valor inteiro superior a v
cos(v)	cosseno de v
exp(v)	e elevado a v
floor(v)	próximo valor inteiro inferior a v
log(v)	logarítimo de v na base 10
max(valor ...)	maior valor da lista
min(valor ...)	menor valor da lista
pow(x, y)	x elevado a y
ramdom()	numero randomico de >= 0 e < 1
round(v)	arredondamento de v
sin(v)	seno de v
sqrt(v)	raiz quadrada de v
tan(v)	tangente de v

- # Objeto String:

Atributos

Nome	Descrição
length	Tamanho da string

Métodos

Nome	Descrição
escape(string)	mesmo que encodeURI
unescape(string)	mesmo que decodeURI

encodeURI(string)	Codificação necessária para passar tring como parametro de URI
decodeURI(string)	Decodificação necessária de string recebida por parametro URI
decodeURIComponent(string)	Decodifica toda a URI passada
encodeURIComponent(string)	Codifica toda a URI passada
anchor(nome)	`new String("sergio").anchor("n");` `retorna: <a name="n">sergio</a>`
big()	`new String("sergio").big();` `retorna: <big>sergio</big>`
blink()	`new String("sergio").blink();` `retorna: <blink>sergio</blink>`
bold()	`new String("sergio").bold();` `retorna: <b>sergio</b>`
charAt(posicao)	o mesmo que substring(posicao, posicao+1), retorna um caractere da string.
charCodeAt(posicao)	Valor unicode do caracter da posição.
concat(item ...)	Concatena itens fornecidos a string.
fixed()	`new String("sergio").fixed();` `retorna: <tt>sergio</tt>`
fontcolor(cor)	`new String("teste").fontcolor("blue"),` `retorna: <font color="blue">teste</font>`
fontsize(tamanho)	`new String("teste").fontsize(16),` `retorna: <font size="16">teste</font>`
indexOf(substring) indexOf(substring, posiçãoInicial)	Procura a ocorrencia da substring dentro da string
italics()	`new String("sergio").italics(),`

	`retorna: <i>sergio</i>`
lastIndexOf(substring) lastIndexOf(substring, posiçãoInicial)	Procura a ultima ocorrência da substring dentro da string.
localeCompare(outraString)	Compara outra string com a string e retorna: negativo se string < outraString 0 se for igual positivo se string > outraString
match(regexp)	retorna posição encontrada segundo expressão regular
replace(valorBusca, valorSubstituição)	troca onde localizar valorBusca por valorSubstituição, o valor de busca pode tanto ser uma string, como uma expressão regular, se for uma string somente a primeira ocorrência será substrituida.
search(valorBusca)	procura por valorBusca
slice(inicio) slice(inicio, fim)	substring do inicio até o fim, fim não é incluso,
small()	`new String("teste").small();` `retorna: <small>teste</small>`
split()	`new String("Sergio Eduardo Rodrigues").split(" ");` `retorna: ["Sergio", "Eduardo", "Rodrigues"]`
strike()	`new String("teste").strike();` `retorna: <strike>teste</strike>`
sub()	`new String("teste").sub();` `retorna: <sub>teste</sub>`
substr(posicao, quantidade)	Retorna a substring da posição inicial, com tamanho fornecido pela quantidade. a posição pode ser negativo, indicando relativo ao final da string.
substring(posiçãoInicial)	Substring da posição inicial até

substring(posiçãoInicial, posiçãoFinal)	posição final, posição final não é inclusa. se a posição final não for fornecida, será retornado a string da posição inicial até fim da string
sup()	new String("teste").sup(); retorna: ^{teste}
toLocaleLowerCase()	Converte para minusculo
toLocaleUpperCase()	Converte para maiusculo
toLowerCase()	Converte para minusculo
toUpperCase()	Converte para maiusculo
<método estático> fromCharCode(numero)	retorna o caracter representado pelo indice unicode <numero>

- # Objeto Number:

Atributos

Nome	Descrição
NaN	Representa valores que não são considerados números
NEGATIVE_INFINITY	Valor infinito negativo
POSITIVE_INFINITY	Valor infinito positivo

Métodos

Nome	Descrição
Number(numero)	Construtor
toString()	Converte para string decimal ou na base fornecida

toString(base)	
toLocaleString()	Converte para string na localidade atual
valueOf()	Converte de um objeto para number
toFixed(decimais)	Retorna string com numero formatado contendo decimais casas.
toExponential(decimais)	Retorna string notação exponencial com decimais dígitos
toPresision(decimais)	Formata numero de dígitos

- # Objeto XMLHttpRequest:

  Com o advento do AJAX, este objeto se torna cada vez mais importante. Ele, infelizmente, não é padronizado, e portanto, até que saia uma resolução, devemos sempre fazer checagem para criação deste objeto, a seguir apresento a sua estrutura, e em seguida uma pequena rotina para criação e manipulação do mesmo.

  Atributos

Nome	Descrição

readyState	Representa o estado do objeto, pode ser :
	`* 0 - nao inicializado (uninitalized)`
	`* 1 - carregando (loading)`
	`* 2 - carregado (loaded)`
	`* 3 - interativo (interactive)`
	`* 4 - completo (complete)`

responseXML	Resposta em xml (document)
responseText	Resposta em texto
status	Valor numero de retorno
statusText	Texto de status
multipart	Indica que esta recebendo um texto multipart

Métodos

Nome	Descrição
stop()	Pára a transferência

getAllResponseReaders()	Retorna nomes dos cabeçalhos
getResponseReader(name)	Retorna valor do cabeçalho
open("metodo", "url"[, indicadorDeAssincrono[, nomeUsuario [, senha]]])	Abre comunicação
send(content)	Envia conteúdo
setRequestHeader("nome", "valor")	Atribui dados a cabeçalho antes do envio
overrideMimeType("mime-type")	sobre escreve o tipo retornado

Eventos

Nome	Descrição
onload	Event listener que recebe event como parametro, assim pode-se receber elementos como resposta
onerror	Evento chamado caso ocorra um erro na carga
onprogress	Evento chamado durante a carga, caso seja um conteudo muito grande para baixar.
onreadystatechange	Evento chamado quando o estado da carga muda.

- Exemplo de Uso:

```
Salve este código como request.js


  var RequestObject;

  function initRequest(newRequestFunc, noBody) {
        var _newRequest = newRequestFunc;
        var _noBody = noBody;
        var _id = 0;
        return function() {
                this.newRequest = _newRequest;
                this.concatTimer = function(url, id) {
                        return url +
                          (url.indexOf("?") < 0 ? "?" : "&")+
                          "requestTime=" + new Date().getTime() +
                          "&requestId=" + id;
                }
                this.loadText = function(url, method) {
                        var req = _newRequest();
                        req.open(method || "GET", this.concatTimer(url, _id++),
false);
                        if (_noBody)
                                req.send();
                        else
                                req.send(null);
                        return req.responseText;
                }
                this.splitLines = function(text) {
```

```
                        try {
                                return text.split(/\r?\n|\r/);
                        } catch(e) {
                                return [];
                        }
                }
                this.loadLines = function(url, method) {
                        return this.splitLines(this.loadText(url, method ||
"GET"));
                }
                this.loadXML = function(url, method) {
                        var req = _newRequest();
                        req.open(method || "GET", this.concatTimer(url, _id++),
false);
                        if (_noBody)
                                req.send();
                        else
                                req.send(null);
                        return req.responseXML;
                }
                this.bind = function(object) {
                        var url = object['url'];
                        if (typeof url == 'undefined')
                                throw "necess?rio URL para fazer bind";
                        var id = _id++;
                        var req = _newRequest();
                        var method = object['method'] || "GET";
                        var headers = object['header'];
                        var body = object['body'];
                        var user = object['username'];
                        var pass = object['password'];
                        var onload = object['onload'];
                        var onerror = object['onerror'];
                        var onprocess = object['onprocess'];
                        var onstatechange = object['onstatechange'];

                        req.onreadystatechange=function() {
                                if (onstatechange)
                                        onstatechange(req, id);
                                switch(req.readyState) {
                                        case 0: // UNINITIALIZED open() não foi
chamado ainda
                                                break;
                                        case 1: // LOADING send() não foi
chamado ainda
                                        case 2: // LOADED send() foi chamado,
disponível getResponseHeader e status
                                        case 3: // INTERACTIVE carregando,
responseText tem dados parciais
                                                if (onprocess)
                                                        onprocess(req, id);
                                                break;
                                        case 4: // COMPLETED, todas as operações
foram concluidas
                                                if (onprocess)
                                                        onprocess(req, id);
                                                if (req.status == 0 ||
req.status == 200) {
                                                        if (onload)
                                                                onload(req, id);
                                                } else {
```

```
                                                        if (onerror)
                                                                onerror(req, id,
req.statusText);
                                                }
                                                break;
                                        }
                                }
                                if (user)
                                        req.open(method, this.concatTimer(url, id),
true, user, password);
                                else
                                        req.open(method, this.concatTimer(url, id));
                                req.setRequestHeader('requestid', id);
                                for(var header in headers) {
                                        req.setRequestHeader(header, headers[header]);
                                }
                                try {
                                        if (body && _noBody) {
                                                req.send();
                                        } else {
                                                req.send(body);
                                        }
                                } catch(e) {
                                        if (onerror) {
                                                onerror(req, id, e);
                                        }
                                }
                        }
                }
        }

        if (window.ActiveXObject) {
                var objectNames = [
                                "Msxml2.XMLHTTP.5.0",
                                "Msxml2.XMLHTTP.4.0",
                                "MSXML2.XMLHTTP.3.0",
                                "MSXML2.XMLHTTP",
                                "Microsoft.XMLHTTP"
                ];
                for(var i=0; i < objectNames.length; i++) {
                        try {
                                var requestName = objectNames[i];
                                new ActiveXObject(requestName);
                                RequestObject = initRequest(function() { return new
ActiveXObject(requestName); }, true);
                        } catch(e) {
                        }
                }
        }

        if (! RequestObject && window.XMLHttpRequest) {
                try {
                        new XMLHttpRequest();
                        RequestObject = initRequest(function() { return new
XMLHttpRequest(); }, false);
                } catch(e) {
                }
        }

        if (! RequestObject) {
                alert("Seu browser não esta preparado para o ajax");
```

```
  }
```

Inclua na página, o seguinte comando para incluir este arquivo :

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

em seguida, você pode utilizar o código, como por exemplo, carregando um texto de forma sincôna :

```
   var req = new RequestObject();
   alert(req.loadText("textoqualer.txt"));
```

podemos também carregar numa matriz e processar linha a linha:

```
   var req = new RequestObject();
   var matriz = req.loadLines("textoqualer.txt");
   for(var i=0; i < matriz.length; i++) {
     alert(i + ":" + matriz[i]);
   }
```

também é possível carregar de forma assincrona :

```
   var req = new RequestObject();
   req.bind({
     url:"textoqualquer.txt",
     onload:function(rq, id) {
       alert(rq.responseText);
     }
   });
```

é possível também verificar se ocorreu algum erro, da seguinte forma :

```
   var req = new RequestObject();
   req.bind({
     url:"textoqualquer.txt",
     onload:function(rq, id) {
       alert(rq.responseText);
     },
     onerror:function(rq, id, msg) {
       alert('erro = ' + msg);
     }
   });
```

- # Outros Objetos:

  Para referências de outros objetos como DOM (Document Object Model), HTML e SVG, visite o capítulo on-line deste livro no site: http://pt.wikibooks.org/wiki/Javascript:_Objetos

CONCLUSÃO:

Espero ter dado uma pequena contribuição para a disseminação desta linguagem script, e possa tornar a vida mais fácil para programadores que necessitam tornar suas páginas mais inteligentes.

REFERÊNCIAS NA INTERNET:

- Definição do Javascript - Definição do javascript pela ecma, orgão europeu de padronização.
- Javascript projeto Mozilla - Referência a tecnologia javascript pelo projeto Mozilla.
- Definição XMLHttpRequest - Definição de XMLHttpRequest pelo Mozilla.

  Javascript Microsoft - Página inicial de javascript na MSDN
- Tutorial de JavaScript - em Português e Inglês
- JScript - Versão da Microsoft do JavaScript, usado no Internet Explorer (Em Inglês).
- JavaScript - Página de JavaScript da fundação Mozilla (Em Inglês).
- Standard ECMA-262 - Especificação oficial do JavaScript (Em Inglês).
- Exemplos do Javascript - no espanhol.
- Guia Javascript da fundação Mozilla um bom guia de javascript em Inglês
- Programação baseada em classe vs. Programação baseada em protótipo essencial para quem está acostumado com o paradigma da programação orientada a objeto, o texto está em Inglês.
- Artigo "Javascript" pela Wikipédia.