JAVASCRIPT

ECMA-262 edição 5.1 (junho de 2011)

(ECMAScript 5 !)

http://www.ecmascript.org/

http://www.ecma-international.org/publications/standards/Ecma-262.htm

Parte 1: Javascript Básica

Parte 2: DOM

Parte 3: AJAX

Esta é uma introdução relâmpago a Javascript *(“crash presentation”*).

Javascript é uma linguagem interpretada, usada principalmente em browsers da Web.

A linguagem foi desenvolvida em somente 10 dias, em maio de 1995, por Brendan Eich, enquanto ele trabalhava para a Netscape.

Ela é uma linguagem multiparadigma, com suporte aos estilos de programação imperativa, funcional e orientada a objetos.

A edição mais nova (edição 6) foi lançada em junho de 2015. Ela possui modificações importantes.

Referências para Javascript:

Site de Mozilla.org (Este tem tudo!):

https://developer.mozilla.org/en-US/docs/Web/JavaScript

Site de Ecmascript (5.1 e outros):

http://www.ecma-international.org/publications/standards/Ecma-262-arch.htm

Site de Ecmascript 6:

http://www.ecma-international.org/publications/standards/Ecma-262.htm

Bons livros para aprender Javascript:

John Resig - "Secrets of the Javascript ninja", Manning Publications, 2009.

Stoyan Stefanov - "Object-Oriented JavaScript: Create scalable, reusable high-quality JavaScript applications,and libraries", Packt Publishing, 2008.

Crockford, Douglas - "Javascript: the good parts", Yahoo! Inc., O'Reilly Media Inc., 2008.

Há várias maneiras de se experimentar Javascript.

Se você já usa o Netbeans para desenvolver aplicações Web, pode criar um projeto Web e uma página com o código html abaixo:

```
<html>
    <head>
        <title>Teste Javascript</title>
        <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />
        <script src="./js/teste01.js"></script>
    </head>

    <body style="background-color:gray;">
        <div style="position:relative;top:100px;border:1px solid black;
                    width:400px; height:200px;margin-left:auto;
                            margin-right:auto;background-color:white;
                            text-align:center;">
                <br />
                <br />
                <br />
                <input type="text" id="hora"
                           style="width:350px;text-align:center;
                                  font-size:14pt;font-weight:bold;"/>
                </div>
        </body>
</html>
```

Além disto, crie o arquivo teste01.js no diretório js, na raiz das WebPages.

Por exemplo:

```
window.onload = executarDepoisDeCarregar;

function executarDepoisDeCarregar(){
        alert('FUNCIONOU!');
        var bool = confirm('FUNCIONOU!');
        var texto = prompt('FUNCIONOU!',"Aqui...");
        console.log("===  "+texto+"  ===");

        document.getElementById("hora").value =
                    new Date().toISOString();
}
```

Agora rode o comando no Netbeans para ver a página html no *browser* (shift-F6 com foco no arquivo html).

Quando você modificar o arquivo teste01.js, basta:

-salvá-lo no Netbeans (cntrl-S),

-mudar o foco para o browser (alt-tab),

-recarregar a página no *browser* (cntrl-R)

Há quatro maneiras simples de interagir com Javascript no ambiente de um *browser*:

1. `alert(“Hello!”);`
2. `var bool = confirm("Confirma?");`
3. `var texto = prompt("Digite","aquí...");`
4. `console.log(“mensagem...”);`

Atenção, pois as três primeiras funções são síncronas e só devem ser usadas junto com funções assíncronas com cuidado.

Você também pode, além disto, utilizar os *plugins* de depuração dos próprios *browsers*.

SÓ ISTO!

Agora vamos começar a ver os elementos de

JAVASCRIPT

Parte 1 : Javascript Básica

1. Palavras Reservadas
2. Variáveis
3. Sortes
4. Números
5. Booleanos
6. Strings
7. Null
8. Undefined
9. Objetos
10. Funções
11. Arrays
12. Comandos
13. Operadores
14. Api básica da linguagem
15. Escopo, contexto e fecho
16. Construtor e new
17. Linking e prototyping

1. Palavras reservadas:

```
abstract
boolean break byte
case catch char class const continue
debugger default delete do double
else enum export extends
false final finally float for function
goto
if implements import in instanceof int
interface
long
native new null
package private protected public
return
short static super switch synchronized
this throw throws transient true try typeof
var volatile void
while with
```

2. Variáveis:

São nomes (identificadores, palavras não reservadas).

São “fracamente tipadas” ou “dinamicamente tipadas”.

Melhor dizendo, NÃO SÃO TIPADAS!

Em cada estado da computação, as variáveis são mapeadas em valores, eventualmente de sortes diferentes.

3. Sortes (conjuntos de *valores*):

Só há dois tipos de sortes em Javascript:

objetos e *undefined*

(A nomenclatura confunde um pouco: podemos dizer que a sorte *undefined* é o conjunto vazio. Mas a palavra reservada da linguagem *undefined* serve para dizer que a variável não tem nenhum valor naquele estado da computação)

3. Sortes:

Há quatro sortes básicas e 2 especiais

1. Boolean
2. Number
3. String
4. Object

- *null*

- *undefined*

Atenção! As 5 primeiras são objetos, isto é, boolean, number, string e null também são objetos!

3. Sortes:

Os objetos de Object ainda podem ser divididos em:

- Definidos pelo programador
- Function (esta sorte de objetos é destacada)
- Array
- Date
- RegExp
- Error types

3. Sortes (dizendo de outra forma...):

- *Sortes primitivas (cujos elementos são objetos!):*
  - Números
  - Booleanos
  - Strings
- *Objetos, definidos pelo programador:*
  - *conjuntos de propriedades (variáveis) e métodos*
- *Objetos pré-fabricados: Date, RegExp, Error, ...*
- *Funções: podem ser primitivas ou definidas pelo programador (funções também são objetos!)*
- Sortes especiais:
  - *null: possui um único elemento (ponteiro nulo, que é um objeto!).*
  - *undefined: não possui nenhum elemento (logo, não é um objeto...).*

QUER DIZER, TODOS OS VALORES (menos undefined) SÃO OBJETOS!

4. Números:

Só tem números de 64 bits, ponto flutuante, IEEE-754 (double de Java).

Inclui, como no padrão IEEE-754, **`NaN`**.

Pode parecer pouco flexível, mas você nunca vai usar o tipo errado de número...

5. Booleanos:

true , false

Valores “falsificantes”, além do *false* (que vão funcionar como se fossem o *false*):

- null
- undefined
- “” (string vazio)
- 0
- NaN

Todos os outros valores são “tautologizantes” (vão funcionar como se fossem o *true*)

6. Strings:

Sequência de 0 ou mais caracteres.

Os caracteres NÃO são valores! Caracteres são elementos léxicos, não sintáticos.

“Caracteres” são strings de comprimento 1.

Os caracteres ECMAScript têm 16 bits (2 bytes).

ECMAScript exterioriza caracteres segundo UCS-2.

(UCS-2 é uma codificação de UNICODE)

A implementação interna costuma ser UCS-2 ou UTF-16.

Aos interessados, sugiro:

http://mathiasbynens.be/notes/javascript-encoding

6. Strings (continuação):

Strings são imutáveis.

Se a e b são strings iguais, caractere a caractere, então a == b avalia igual a *true*.

Literais de string podem ser escritos como

‘string’ ou “string”

(apóstrofos ou aspas, como diz o povo:

aspas simples ou aspas duplas).

7. null:

- Usado para significar “não possui um outro valor”.
- Pode (e deve) ser usado pelo programador.
- É um objeto.

8. undefined:

- Usado para significar “não foi iniciado” ou “não existe”.
- É usado pelo interpretador JS.
- É o valor default de variáveis, de argumentos de funções e de membros inexistentes de objetos.
- É o único valor que NÃO é um objeto! Melhor dizendo, *undefined* não é um valor, é a ausência de um valor. É o elemento de um conjunto vazio, isto é, não existe.

9. Objetos:

Declaração: notação literal, com chaves

```
var meuObjeto = {  ... }
```

São conjuntos de pares “nome:valor”, separados por vírgulas.

O nome é um identificador (também chamado de propriedade).

O valor é um elemento de qualquer uma das sortes já apresentadas (o valor da propriedade).

```
var meuObjeto = {nivel:5 , mensagem:“ERRO”}
```

(esta é a *Javascript Standard Object Notation* - JSON)

9. Objetos (continuação):

Na verdade, como os valores primitivos também são objetos e as funções também são objetos, um objeto definido pelo programador, com a ajuda das chaves {...}, é um HashMap de outros objetos.

Por isto, objetos também são chamados de “arrays associativos”.

Outro exemplo:

```
var objetoExemplo = {
            nivel:“URGENTE”,
            mensagem:“ERRO”,
            reconhecer:function(){alert(‘ERRO!’);}
        }
```

9. Objetos (continuação):

Para se listar os nomes de todas as propriedades de um objeto, há a função keys de Object. Esta função devolve um array com os nomes.

```
var obj =
   {zeroN:"zeroV", umN:"umV", doisN:"doisV"};

var keys =  Object.keys(obj) ;

for(i=0;i<keys.length;i++){
  console.log(console.log(
                keys[i]+' : '+obj[keys[i]] );
}
```

9. Objetos (continuação):

Os objetos em Javascript podem ser dinamicamente aumentados:

```
var obj1 = {};
obj1.propriedadeA = ‘Esta é nova’;
Obj1.metodoB = function(){
                    // tambem vale para funcoes
                    ... };
```

Isto pode ser verificado com o método keys() de Object.

10. Funções:

São objetos destacados, com uma sintaxe especial!

Declaração:

```
var minhaFuncao = function (){  ...
}
```

que é equivalente a

```
function minhaFuncao(){ ... }
```

A palavra reservada *function* com *()* significa que o objeto {...} pode ser invocado!

10. Funções (continuação):

Como pode ser observado na primeira declaração da função, o lado direito do sinal de igual define uma função sem nome, como são todos os objetos.

Este estilo de tratar uma função sem nome é conhecido como a *notação lambda*.

Uma função é então um valor (objeto) como qualquer outro, podendo ser

- atribuído a variáveis,
- passado como argumento ou
- retornado.

10. Funções (continuação):

Maneiras de invocar uma função:

Pelo nome:

```
functionObject(arguments)
```

Como método:

```
thisObject.methodName(arguments)
thisObject["methodName"](arguments)
```

Pelo construtor:

```
new functionObject(arguments)
```

(continua)

10. Funções (continuação):

Maneiras de invocar uma função (continuação):

Usando o método *apply*:

```
functionObject.apply(scopeObj,argsArray)
```

Usando o método *call*:

```
functionObject.call(scopeObj,argsList)
```

11. Arrays:

Arrays são objetos com uma notação especial que torna o seu uso mais parecido com o uso nas outras linguagens como C e Java.

Basicamente, pode-se declarar um array da seguinte forma:

```
var meuArray = [];
```

Ao invés da declaração usual de objetos:

```
var meuObjeto = {};
```

11. Arrays(continuação):

A declaração de um array desta forma cria um objeto com uma propriedade a mais chamada *length*.

O acesso às componentes do array é feito da forma usual:

```
var valor = meuArray[5];
```

O interpretador JS transforma automática e transparentemente este acesso no seguinte acesso:

```
var valor = meuArray[“5”];
```

11. Arrays(continuação):

A adição de novas componentes ao array aumenta automaticamente o valor de *length*.

Como se trata na verdade de um objeto, acessos a componentes inexistentes vão produzir resultados *undefined*.

Esta definição de arrays é particularmente eficiente para tratar de arrays esparsos.

11. Arrays(continuação):

Há dois métodos simples, mas muito básicos e úteis, de Array:

```
var numeros = new Array(5,6,7);

numeros.unshift(1,2,3,4);
                    // [1,2,3,4,5,6,7]

numeros.push(8,9);
                    // [1,2,3,4,5,6,7,8,9]
```

12. Comandos:

Javascript possui os comandos usuais de outras linguagens, como C e Java:

atribuição,

Expressões e operadores,

if, switch, for, while, do,

Labels, break, continue,

return,

try/catch/throw

13. Operadores:

Javascript possui um bom número de operadores, à semelhança de outras linguagens como C e Java:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators

13. Operadores (continuação):

O operador prefixado **typeof** retorna uma string que identifica a sorte do valor.

sorte	retorno de typeof
object	**'object'**
function	**'function'**
array	**'object'**
number	**'number'**
string	**'string'**
boolean	**'boolean'**
null	**'object'**
undefined	**'undefined'**

13. Operadores (continuação):

Podemos observar **quatro** coisas acerca do operador **typeof**:

O retorno para um array é *object*, como se podia esperar.

Mas o retorno para uma função é *function*, embora ela também seja um objeto!

O retorno de *null* é *object* (sim, *null* é um objeto!)

O retorno de *undefined* é *undefined*, isto é: undefined não é nem um objeto!

14. API básica da linguagem:

Javascript possui uma extensa coleção de funções e objetos básicos da linguagem com seus métodos.

Uma descrição resumida destas funções e objetos tomaria um espaço mais de dez vezes o destas notas.

Sugiro:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

15. Escopo, Contexto e Fecho (*Closure*):

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures

Uma boa apresentação sobre escopo, contexto e fecho:

http://ryanmorr.com/understanding-scope-and-context-in-javascript/

ou sua tradução para português por J.L.S.Leão

16. Construtor e new:

Javascript permite criar novos objetos a partir de um objeto **definido como uma função**, através o uso do operador new:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects

17. Linking e prototyping:

A orientação a objeto de Javascript é baseada em prototyping (não em classes)

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain

Parte 2 : DOM

Ver:

Introducing the JavaScript DOM

http://www.elated.com/articles/javascript-dom-intro/

Parte 3 : AJAX

Exemplo de código Javascript para um pedido AJAX:

```
function fazerPedidoAJAX(argumento){
   // Criar o novo objeto XMLHttpRequest (o objeto do AJAX)
   var ajaxRequest = new XMLHttpRequest();
   // Definir o metodo e o destino
   ajaxRequest.open(
                    "GET“,
                    "http://localhost:8084/ajaxBemSimples/controller?nomeDoParametro=" + argumento
    );
   // Definir o Content-type do pedido HTTP
   ajaxRequest.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
   // Preparar o recebimento da resposta (definir funcao de callback)
   ajaxRequest.onreadystatechange =
         function() {
            if(ajaxRequest.readyState===4 && ajaxRequest.status===200){
               document.getElementById("IDdadosResposta").value = ajaxRequest.responseText;
            }
         };
   // Enviar o pedido
   ajaxRequest.send(null);
}
```