JavaScript – Resumo

Referência : https://www.w3schools.com clicar em References

Notas sobre o funcionamento do JavaScript dentro dos browsers.

1-O JavaScript acessa os elementos do browser via Object Document Model, ou seja, um modelo de objetos do browser. Sempre que temos:

document.xxxx
estaremos nos referindo ao dom

2-Quando colocamos o JavaScript no Header da página este será 'compilado' antes dos elementos da página serem renderizados, ou seja, o script é compilado (e executado) antes dos elementos HTML da página serem definidos.

Portanto, ao colocarmos um script no Header da página este não poderá ter uma referência a um objeto do DOM porque ele ainda não existirá no momento.

Por exemplo, uma página:

```
<html>
    <head>
        <meta charset="utf-8">
        <title>JavaScript</title>
        <script>
            document.getElementById('elem1').value = 'Oi';
        </script>

    </head>

    <body>
    <h1>Curso de JavaScript</h1>
        <input  type="text"  placeholder="Teste de precedência"  id="elem1"
        disabled="disabled"  >
    </body>
</html>
```

Vai dar erro de elemento ainda não definido.

3- ***Javascript é Case Sensitive.***

Então 'Nome' é diferente de 'nome'

Resumo – Início

1-Comentários

Comentários são muito úteis para documentar uma condição / situação / funcionalidade. Isto evita possibilidades de erro de interpretação bem como facilita os processos porque não será necessário analisar o código para saber o que ele faz.

// define que o que estiver a direita deste símbolo é comentário e não deve ser entendido como código JavaScript.

```
/*
comentário
```

/ define que as linhas entre / e */ são comentários

2-Variáveis

São posições de memória que armazenam informações que podemos alterar ou recuperar como desejarmos.
Toda variável em javascript é um objeto e na atribuição define-se tipo, valor.

2.1-Tipos de variáveis em JavaScript

String	cadeia de caracteres, textos
number	numérica – pode ser tanto inteira, fracionária, ponto flutuante
boolean	armazenam apenas verdadeiro ou falso. Úteis para testes condicionais.

2.2-Regras para nomes das variáveis

2.2.1-Não podem ser iniciados com números, apenas letras ou sublinhado.

2.2.2-Não podem ter caracteres especiais (ç, ^, ~) ou acentuados(á,é,í,ó,ú)
Importante : espaço é um caractere especial em JavaScript.

2.2.3-Não podem ser iguais as palavras reservadas em JavaScript.
abstract, arguments, await*, boolean, break, byte, case, catch,
char, class*, const, continue, debugger, default, delete, do,
double, else, enum*, eval, export*, extends*, false, final,
finally, float, for, function, goto, if, implements, import*,
in, instanceof, int, interface, let*, long, native, new,
null, package, private, protected, public, return, short, static,
super*, switch, synchronized, this, throw, throws, transient, true,
try, typeof, var, void, volatile, while, with, yield

2.3-Como as variáveis são declaradas

var nomevariavel	var NomePessoa	ok
var 200nomes	var número	errados

Importante:

1 - ***É no momento da atribuição de valor que o tipo de variável é definido***.

Portanto se	num vb é	dim a as integer = 2
	num c# é	int a =2
	no javascript é	var a = 2

Nota: o usuário não pode definir o tipo da variável, é um processo automático.

Note que uma variável pode ter seu tipo mudado no decorrer do processo:

var a='Teste'	a=5
document.write(a);	escreverá 5 na tela

2- ***O Javascript utiliza a notação americana para variáveis numéricas***.

Portanto, o ponto decimal que para nós é uma vírgula para o JavaScript é o Ponto.
Não existe o separador de milhar na atribuição (mas na exibição pode ser formatado assim).
Ex: var a = 2.17189

3-Muitas pessoas colocam um sufixo antes do tipo da variável para facilitar a depuração e evitar enganos com as operações dessas variáveis. É boa pratica colocar esses prefixos:

Tipo da variável	prefixo
String	str
Number	nbr
Boolean	bol

Exemplos : var intA = 2; var strNome = 'Antonio'

4-***A atribuição de valores em Javascript é feito com o sinal de igual simples***

Ex.: var strTexto = 'Texto' var Texto2 = "texto"
var nbrX = 2 var nbrPi = 3.1415926535
var intJ = -7

Importante:

1-Para fazer o teste de igualdade se usa dois sinais de igual juntos.
Ex: if(a == 2){}

2-Não esquecer de fechar a aspas (simples ou duplas) nas atribuições de string.
O JavaScript se perde completamente se esquecermos disso.

3-Document.write

Escreve um texto ou valor de variável no browser
Ex.: document.write("bom dia")

4-Concatenação (+)

Esta operação ocorre quando somamos strings, textos.
Ex.: var a='Hoje é' var b= 'Quinta-Feira'
document.write(a+' '+b)
Vai escrever...Hoje é Quinta-Feira.

Nota: sempre que estamos somando string com qualquer outro tipo de variável o valor da variável não string é convertido para string.

Ex: var Nome='José'
Var Idade =33
Document.write("Bem vindo " + Nome + ", hoje você tem " + Idade + " anos")
Vai exibir: Bem vindo José, hoje você tem 33 anos.

Importante: O mesmo operando de soma serve para somar números.
Portanto se tivermos o seguinte código:
var a = 5; var b = 7;
console.log(a+b); irá retornar 12

Se a soma for mista de tipos sempre será utilizado o tipo string.
Portanto se tivermos o seguinte código:
var a = ‘5’; var b = 7;
console.log(a+b); irá retornar 57

5-Prompt

Permite fazer uma pergunta ao usuário sobre um valor.
Sempre será recebido um texto.
Ex.: var nome = prompt(“Digite o seu nome”)

6-Null

É a ausência intencional de um valor. A variável foi declarada e tem um valor vazio, sem significado.
Neste caso a variável ainda é um objeto.

```
Ex.: <doctype html>
<html>
	<head>
	<meta charset="utf-8">
	<script>

		var teste1 = null;
		document.write('Tipo da variável: ' + typeof(teste1));		//object

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

7-Undefined

Define que uma variável foi declarada mas não foi feita nenhuma atribuição até o momento e por este motivo seu tipo continua indefinido.

```
Ex.: <doctype html>
<html>
	<head>
	<meta charset="utf-8">
	<script>

		var teste2;		//ou var teste2=undefined;
		document.write('Tipo da variável: ' + typeof(teste2));		//undefined

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

8-Operadores de comparação ou operadores relacionais

Em javascript temos os seguintes operadores (num total de 8):

Operador	operação	processo
==	igual	verifica se os valores comparados são iguais.
===	idêntico	verifica se os valores comparados são iguais e do mesmo tipo.
!=	diferente	verifica se os valores comparados são diferentes.
!==	não idênticos	verifica se os valores e os tipos comparados são diferentes.
<	menor	verifica se o valor da esquerda é menor que o da direita
>	maior	verifica se o valor da esquerda é maior que o da direita
<=	menor ou igual	verifica se o valor da esquerda é menor ou igual ao da direita
>=	maior ou igual	verifica se o valor da esquerda é maior ou igual ao da direita

8.1-Comparação

A comparação de valores primitivos é a seguinte:

Equality(==)	null	undefined	false	""	0	NaN
null	=	=	≠	≠	≠	≠
undefined	=	=	≠	≠	≠	≠
false	≠	≠	=	=	=	≠
""	≠	≠	=	=	=	≠
0	≠	≠	=	=	=	≠
NaN	≠	≠	≠	≠	≠	≠

Quando fazemos a comparação entre um variável tipo string e uma variável numérica, se o valor da string for um número, este será convertido para número e depois será feita a comparação. Portanto:

If('5'==5){} é uma expressão que retorna verdadeiro
var a = 5 var b = '5'
if(a==b){} é uma expressão que retorna verdadeiro

9-Conversão de tipos

9.1-toString()

Converte um valor numérico em string.

```
document.write("Definindo duas variaveis numéricas e somando. 7 + 8=")
var l = 7;
var m = 8;
document.write(l+m);                    //exibe 15

document.write("<br>")
```

```
document.write("Convertendo para string e somando(7+8)=")
l=l.toString();
m=m.toString();
document.write(l+m);                    //exibe 78
```

9.2-parseInt

Converte de String para Inteiro desprezando a parte fracionária se houver.

```
document.write("De string('7') para numérico(int)+3=")
var i = '7';            ou          var i = '7.98';
document.write(parseInt(i)+3);      exibe 10
```

9.3parseFloat()

Converte um string para número mantendo a parte fracionaria.

```
document.write("De string('7.98') para numérico(float)+3=")
var k = '7.98';
document.write(parseFloat(k)+3);            exibe 10.98
```

10-Operadores lógicos

Permitem agrupar um conjunto de condições de maneira a realizar as necessidades do mundo real.

10.1-&& - And (e) Lógico

Retorna true se ambos os operandos forem verdadeiros.
Usamos quando duas ou mais condições tem que ser, todas, verdadeiras.

Ex.: Para se alistar no exército precisa ser maior de 18 anos **e** ser brasileiro nato.

10.2-|| - Or Lógico

Retorna true se um dos operandos forem verdadeiros.
Usamos quando basta que apenas uma condição seja verdadeira.

Ex.: Ele é baiano **ou** paranaense.

Estas condições são mutuamente excludentes (ou exclusivo)
Ou ele nasceu no estado ou no outro ... não dá pra nascer em 2 estados.

Ex.: Ele pode pagar hoje **ou** amanhã.

10.3-! – Negação

O sinal de exclamação inverte o que vem logo a seguir.
Por exemplo : !(false)=true !(true)=false

```
Ex.:    var a =2            var b = 3
        If (a == b){}       retorna falso ... 2 <> 3
        If (!(a == b)){}    retorna true ... (2 <> 3)=false e !(false)=true
```

11-Operador ternário

Permite fazer uma decisão de maneira mais enxuta, simples mas tem seu uso mais restrito porque não permite agrupar mais de um comando como resultado.

```
Uso:    var resultado = condição ? verdadeiro   : falso
        Verdadeiro é o resultado que irá retornar se a condição for verdadeira
        Falso é o resultado que retornará se a condição for falsa
```

```
Ex.:    document.write("Operador ternário<br>");
        document.write("   uso :      var resultado = condição ? verdadeiro        :
falso<br>");
        var nota = 7;
        var media = 7;
        var faltas = 2;
        var faltas_maximas=5;

        document.write("Nota:" + nota + ", Media:" + media + ", Faltas:" + faltas + ",
MaxFaltas:" + faltas_maximas + ". Resultado=");
        var resultado = ((nota >= media) && (faltas <= faltas_maximas)) ? 'Aprovado' :
'Reprovado';
        document.write(resultado+"<br>");      //exibe Aprovado

        var faltas = 7;
        document.write("Nota:" + nota + ", Media:" + media + ", Faltas:" + faltas + ",
MaxFaltas:" + faltas_maximas + ". Resultado=");
        var resultado = ((nota >= media) && (faltas <= faltas_maximas)) ? 'Aprovado' :
'Reprovado';
        document.write(resultado+"<br>");                //exibe Reprovado
```

12-Switch

Permite agrupar um monte de condições num único comando

```
Uso :           Switch(opção){
                    case valor1:
                        //comandos
                        break;
                        ....
                    default:
                        //comandos
                        break;
                }
Ex.:
        var parametro = 2;

        document.write("Parametro = " + parametro + " : Resultado = ");
        switch(parametro){
            case 1 :
                document.write("É o valor 1");
                break;
            case 2 :
                document.write("É o valor 2");
                break;
            case 3 :
                document.write("É o valor 3");
                break;
            default:
                document.write("Default-Nenhum dos valores");
                break;
```

```
}
```

Importante: O switch leva em conta o tipo da variável. Portanto, 2 é diferente de '2'.
Mude o valor do exemplo acima de 2 para '2' e vai cair no default.

13-Operadores aritméticos

São operadores matemáticos que nos permitem efetuar cálculos básicos.

Nota : Todos os operadores aritméticos caso recebam um string como parâmetro eles automaticamente convertem o valor para numérico, EXCETO a SOMA(+) que pode somar valores numéricos ou concatenar (adicionar) dois strings.

São eles:

13.1-Adição (+)

Soma de valores numéricos.

```
document.write("+ : Somando dois números :<br>");
var a=9;                    var b=2;
document.write("a="+ a + ";" + p + "b=" + b + "; " + p + "a + b = ")
document.write(a + b)
document.write("<br><br>");
```

Importante: Ao somar um número ou string com uma variável não definida ainda teremos como retorno **NAN**:

Ex.:
```
document.write("+ : Somando um número com undefined:<br>");
var a=9;                    var b=undefined;
document.write("a="+ a + ";" + p + "b=" + b + "; " + p + "a + b = <b>")
document.write(a + b)                    //exibe Not a Number (NaN)
document.write("</b><br><br>");
```

13.2-Subtração (-)

Subtraí dois valores numéricos.

```
document.write("- : Subtrai dois números<br>");
var a=9;                    var b=2;
document.write("a="+ a + ";" + p + "b=" + b + "; " + p + "a - b = ")
document.write(a - b)
document.write("<br><br>");
```

13.3-Multiplicação (*)

Multiplica dois valores numéricos

```
document.write("* : Multiplica dois números<br>");
var a=9;                    var b=2;
document.write("a="+ a + ";" + p + "b=" + b + "; " + p + "a * b = ")
document.write(a * b)
document.write("<br>");
```

13.4-Divisão (/)

Divide dois valores numéricos

```
document.write("/ : Divide dois números<br>");
```

```
var a=9;                    var b=2;
document.write("a="+ a + ";" + p + "b=" + b + "; " + p + "a / b = ")
document.write(a / b)
document.write("<br>");
```

13.5-Módulo (%)

Retorna o resto da divisão de dois valores numéricos

```
document.write("% : Módulo ou Resto da Divisão Inteira<br>");
var a=9;                    var b=2;
document.write("a="+ a + ";" + p + "b=" + b + "; " + p + "a % b = ")
document.write(a % b)
document.write("<br>");
```

13.6-Incremento (++)

Soma 1 ao valor da variável atual

Nota: `var a = 1;` `var b=a++;` `var c = ++a;`
A variável b terá o valor 2 e a variável c terá o valor 1

Ex.:

```
document.write("var b = a++;"+p+"faz a atribuição antes do incremento<br>");
document.write("var b = ++a;"+p+"faz o incremento antes da atribuição<br>");

var a = 1;
document.write("var a="+ a +";" + p)
var b = a++;
document.write("var b=a++;" + p + "Valor de b=" + b +p);          //a=2,b=1
document.write("Valor de a=" + a +"<br>");

var a = 1;
document.write("var a="+ a +";"+p)
var b = ++a;
document.write("var b=++a;"+p+"Valor de b=" + b +p);  //a=2,b=2 -
document.write("Valor de a=" + a +"<br><br>");
```

13.7-Decremento (--)

Subtrai 1 ao valor da variável atual

```
document.write("var b = a--;"+p+"faz a atribuição antes do decremento<br>");
document.write("var b = --a;"+p+"faz o decremento antes da atribuição<br>");
```

Ex.:

```
var a = 1;
document.write("var a="+ a +";"+p)
var b = a--;
document.write("var b=a--;"+p+"Valor de b=" + b +p);   //b=1, a=0
document.write("Valor de a=" + a +"<br>");

var a = 1;
document.write("var a="+ a +";"+p)
var b = --a;
document.write("var b=--a;"+p+"Valor de b=" + b +p);   //b=0, a=0
document.write("Valor de a=" + a +"<br>");
```

13.8 – Precedência de Operadores

Quando montamos uma expressão o JavaScript segue os seguintes critérios para sua resolução:

A tabela seguinte está ordenada da mais alta (20) para a mais baixa (0) precedência.

Precedência	**Tipo do Operador**	**Associatividade**	**Operadores individuais**
20	Agrupamento	n/a	(…)
19	Acesso a Membro	esquerda para direita	… . …
	Acesso a Membro Computado	esquerda para direita	… […]
	new (com lista de argumentos)	n/a	new … (…)
18	Chamada a Função	esquerda para direita	… (…)
	new (sem lista de argumentos)	direita para esquerda	new …
17	Incremento Pós-fixado	n/a	… ++
	Decremento Pós-fixado	n/a	… --
16	NÃO lógico	direita para esquerda	! …
	NÃO bit-a-bit	direita para esquerda	~ …
	Positivo Unário	direita para esquerda	+ …

	Negativo Unário	direita para esquerda	- …
	Incremento Pré-fixado	direita para esquerda	++ …
	Decremento Pré-fixado	direita para esquerda	-- …
	typeof	direita para esquerda	typeof …
	void	direita para esquerda	void …
	delete	direita para esquerda	delete …
15	Exponenciação	direita para esquerda	… ** …
14	Multiplicação	esquerda para direita	… * …
	Divisão	esquerda para direita	… / …
	Resto	esquerda para direita	… % …
13	Adição	esquerda para direita	… + …
	Subtração	esquerda para direita	… - …
12	Deslocamento de bits para esquerda	esquerda para direita	… << …
	Deslocamento de bits para direita	esquerda para direita	… >> …
	Deslocamento de bits para direita, sem sinal	esquerda para direita	… >>> …
11	Menor Que	esquerda para direita	… < …
	Menor ou Igual a	esquerda para direita	… <= …
	Maior Que	esquerda para direita	… > …
	Maior ou Igual a	esquerda para direita	… >= …

	in	esquerda para direita	… in …
	instanceof	esquerda para direita	… instanceof …
10	Igualdade	esquerda para direita	… == …
	Desigualdade	esquerda para direita	… != …
	Igualdade Estrita	esquerda para direita	… === …
	Desigualdade Estrita	esquerda para direita	… !== …
9	E bit-a-bit	esquerda para direita	… & …
8	OU exclusivo bit-a-bit	esquerda para direita	… ^ …
7	OU bit-a-bit	esquerda para direita	… | …
6	E lógico	esquerda para direita	… && …
5	OU lógico	esquerda para direita	… || …
4	Condicional	direita para esquerda	… ? … : …
3	Atribuição	direita para esquerda	… = …
			… += …
			… -= …
			… *= …
			… /= …
			… %= …
			… <<= …
			… >>= …
			… >>>= …
			… &= …
			… ^= …
			… |= …
2	yield	direita para esquerda	yield …
	yield*		yield* …
1	Propagação	n/a	… …
0	Vírgula / Sequência	esquerda para direita	… , …

Nota : Memorizar a lista acima é difícil. Normalmente utilizamos ***parênteses*** para darmos prioridade para que uma operação seja feita antes da outra.

Exemplo:

```
document.write("<b>Precedência de Operadores</b><br>");
document.write("O Normal é de de cima para baixo e da esquerda para a direita<br>");
document.write(p+"Mas dependendo da precedencia pode mudar<br>");
document.write(p+"Na dúvida utilize parenteses para alterar a precedência<br><br>");
document.write("Expressão 1:" + p + " 2 + 3 * 2 " + p + "Resulta em ");
document.write(2 + 3 * 2)          //      exibe 8
document.write(p+"Multiplicou 3 * 2 e ai somou 2<br><br>");

document.write("Expressão 1:" + p + " (2 + 3) * 2 " + p + "Resulta em ");
document.write((2 + 3) * 2)                //exibe 10
document.write(p+"somou 2 + 3 e ai multiplicou por 2<br><br>");
```

14-Funções

-Encapsula um bloco de código com funções bem definidas.
-É um subprograma dentro do script
-O nome da função deve seguir as mesmas regras das variáveis (veja item 2.2)
-Costuma-se compor o nome da função como um verbo no infinitivo (indicando o que vai fazer) e um substantivo (para quem vai fazer) com apenas a primeira letra maiúscula.
-Pode-se passar nenhum a quantos parâmetros desejarmos para uma função
-Pode retornar ou não (void) alguma informação.
-Os parâmetros passados para uma função são sempre separados por vírgulas.

Importante: O interpretador JavaScript primeiramente carrega as funções e depois executa os scripts. Portanto, podemos ter a chamada de uma função antes de sua declaração da função que não ocorrerá erro.

Ex.:

```
document.write("<b>Exemplo de uma função</b><br>");

function calcularAreaRetangulo(largura,comprimento){
    var area = largura * comprimento;
    return area;
}
document.write("Chamando a função : calculaAreaRetangulo(8,30) temos o resultado = ");
document.write(calcularAreaRetangulo(8,30));
```

Importante: Diferentemente de outras linguagens o JavaScript é bem maleável em relação as funções. Por exemplo, em VBNET e C# se você coloca 2 parâmetros numa função /sub a sua chamada tem que ter os 2 parâmetros (a menos que você coloque como optional) senão irá provocar erro de compilação. No JavaScript isso não acontece :

1-Se houver parâmetros a mais a função despreza os parâmetros que sobraram. Não há erro.

2-Se houver parâmetros a menos o JavaScript coloca o valor undefined no parâmetro e faz as operações com o valor undefined.

Lembre-se que undefined operação qualquer coisa
Retorna NaN (not a Number – Não é um número)

15-Escopo de variáveis

Temos 3 escopos possíveis de variáveis.

15.1-Global

A variável criada neste escopo será visível em todas as partes do script.

```
<doctype html>
<html>
	<head>
		<meta charset="utf-8">

		<script>
			//Escopo global
			var Serie1 = 'Friends';
			document.write("Escopo Global-Serie1="+Serie1+"<br>");
		</script>
	</head>
	<body style="font-family:Calibri">
	</body>
</html>
```

15.2-Bloco

É quando uma variável é definida dentro de um escopo e, por um detalhe do JavaScript, seu escopo é promovido. Este efeito chama-se ***hoisting***).
Dentro do JavaScript temos 3 escopos: O global, o de função e o de bloco.
Quando uma variável é definida com var dentro de uma função ela pode ser promovida para o escopo global.
Quando uma variável é definida com var dentro de um bloco ({}) ela pode ser promovida para o escopo global (se o bloco estiver fora de uma função) ou para o escopo de função se o bloco estiver dentro de uma função.

Nota: Caso não deseje que esta elevação seja feita (para isolar as variáveis globais com as de bloco) use o comando **let** ao invés de **var**:
Ex.: var a = 'Teste' let a='Teste'

```
<doctype html>
<html>
	<head>
		<meta charset="utf-8">

		<script>
			if(true){
```

```
                var Serie2 = 'Game of Thrones';
                document.write("Escopo Bloco-Serie2="+Serie2+"<br>");
            }
            document.write("Escopo fora da function x-Serie2="+Serie2+"<br>");
        </script>
    </head>
    <body style="font-family:Calibri">
    </body>
</html>
```

15.3-Escopo local

É quando a variável é definida dentro de um container e fica visível apenas dentro deste container :

```
<html>
    <head>
        <meta charset="utf-8">

        <script>
            function x(){
                var Serie3 = 'Walking Dead';
                document.write("Escopo function x-Serie3="+Serie3+"<br>");
                //exibe
            }
            document.write("Escopo fora da function x-Serie3="+Serie3+"<br>");
            //não exibe
        </script>
    </head>
    <body style="font-family:Calibri">
    </body>
</html>
```

16-Hoisting

É a funcionalidade do interpretador javascript de trazer as declarações para o topo do script, ou seja, primeiramente o javascript procura as declarações de variáveis, funções, etc. e depois executa o script. Com isto, por exemplo, podemos usar uma variável antes de sua declaração.

Importante: Testei no IE11 e no Edge e não funcionou.

17- Funções anônimas e a técnica de wrapper (Curso Udemy - item 175)

Como uma variável javascript é um objeto ela permite encapsular até uma função dentro de seu conteúdo.

Na realidade a função não é anônima porque seria inútil se fosse, ela é encapsulada por uma variável.

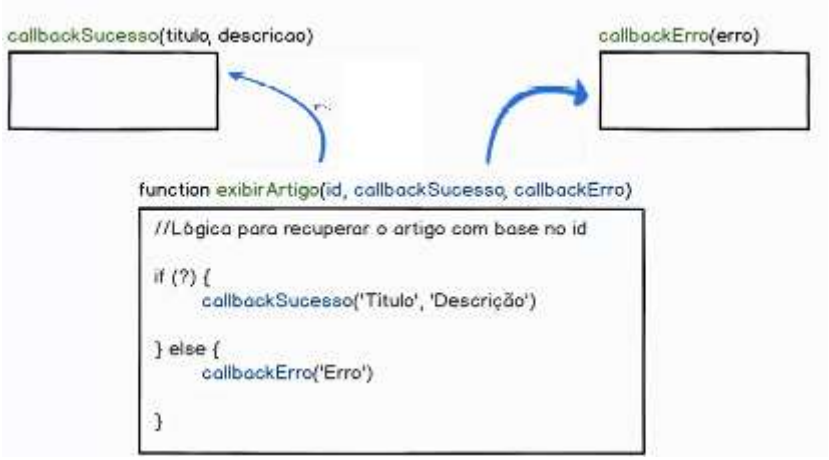
```
//declaração
var encapsula = function(){
    document.write(“OLA”);
}

encapsula();            //chamada
```

Nota: Esta funcionalidade é muito usada no Javascript quando uma função chama outra função.

18-Funções de call-back

São funções que recebem como parâmetros o nome de outras funções e decide por meio de algum processamento qual função, recebida por parâmetro, deverá ser chamada. É o caso de uma função chamar outra função passando parâmetros e o nome de 2 outras funções, uma para caso de sucesso e outra para o caso de erro.



19- Strings

Referência: https://www.w3schools.com/jsref/jsref_obj_string.asp

Notas:
1-Toda string começa da posição zero.
2-Algumas funções são novas e não funcionam em browsers antigos (Inclusive IE11)
3-As strings podem ser declaradas com apóstrofo e com aspas.

Mas se começamos com aspas devemos terminar com aspas, se começamos com apóstrofo devemos terminar com apóstrofo.

Podemos incluir dentro da string aspas ou apóstrofos sem que sejam da declaração de seu início ou fim. Ex.: 'Meu carro é um "Fusca verde" quase "limão"'
4-O browser formata a exibição de algumas strings.

Por exemplo: Um monte de espaços ele exibe apenas um.
Caracteres especiais muitas vezes sequer são exibidos (CR, LF)

5-Caso seja necessário o javascript tem uma representação especial para os caracteres especiais. São eles:

Caractere	Representação	Equivalência ASCII
NULL	\0	ASCII(0)
BACKSPACE (BS)	\b	ASCII(8)
TAB	\t	ASCII(9)
LINE FEED (LF)	\n	ASCII(10)
LINE TABULATION (VT)	\v	ASCII(11)
FORM FEED (FF)	\f	ASCII(12)
CARRIAGE RETURN (CR)	\r	ASCII(13)
QUOTATION MARK (aspas)	\"	ASCII(34) (*1)
APOSTROPHE (apóstrofo)	\'	ASCII(39) (*1)
REVERSE SOLIDUS (barra inv.)	\	ASCII(92) (*1)

*1: Estes caracteres tem uso especial em JavaScript não apenas como caracteres.

19.1 – Propriedades dos Strings

Nota: No arquivo Funções de String.html tem uma descrição detalhada de como usar as funções. Retirei o código porque estendeu muito o número de páginas deste arquivo.
Além disso muitas propriedades, critérios e detalhes foram retirados deste documento para que não ficasse muito extenso.

19.1.1-constructor: Retorna o tipo da função que criou (construtora) o objeto

19.1.2- length : Retorna o tamanho (número de caracteres) do string

19.1.3-prototype: Permite criar uma nova propriedade em um item já existente.

19.2 – Métodos para Strings

Importante: Todos os métodos retornam um novo valor. Eles não mudam a variável ou string original.

19.2.1-charAt() - Retorna o caractere do string orignal que ocupa a posição determinada pelo parâmetro recebido.

Importante: O último caractere de um string é string.length-1

19.2.2-charCodeAt() -Retorna o Unicode do caractere do string especificada pelo índice recebido por parâmetro.

19.2.3-concat() - Une dois ou mais strings e retorna a união deles.

19.2.3-endsWith() - Verifica se a string termina com o(s) string /caracteres especificados
Retorna true se SIM e false se não.
Importante: Só funciona do IE12 em diante.

19.2.4-fromCharCode()
Converte os códigos de caracteres Unicode em caracteres

Uso: String.fromCharCode(num)

Exemplo :

```
var res = String.fromCharCode(65);
res = res+ String.fromCharCode(66);
document.write(res);
```

Retorna : AB

19.2.5-includes() - Verifica se o string contém o(s) string / caracteres especificados
Importante:Não funciona no IE11

19.2.6-indexOf() - Retorna a posição da primeira ocorrência de um valor numa string

19.2.7-lastIndexOf() - Retorna a última posição da ocorrência de um valor dentro do string.
Procura da direita para a esquerda da string

19.2.8-localeCompare() - Compara dois string de acordo com o idioma (current locale)
O idioma é definido no browser (language settings)

19.2.9-match() - Verifica se o string casa contra uma expressão regular e retorna os acertos.
Cada um desses acertos virá separados por vírgula.

19.2.10-repeat() - Retorna um novo string com o número de cópias pedidas do string enviado

19.2.11-replace() - Procura no string um determinado valor ou expressão regular
e retorna uma nova string onde os valores especificados são substituídos

19.2.12-search() - Procura no string por um valor especificado ou expressão regular
e retorna a posição se ela for encontrada

19.2.13-slice() - Extrai uma parte de um string e retorna num novo string

19.2.14-split() - Fatia um string numa matriz com as partes desse string

19.2.15- startsWith() - Verifica se um string (str1) inicia com os caracteres especificados(str2)

19.2.16-substr() - Extrai caracteres de um string, começando de uma posição determinada e
uma quantidade determinada de caracteres

19.2.17- substring() - Extrai os caracteres do string entre os índices especificados

19.2.18-toLocaleLowerCase() - Converte o string para minúsculas de acordo como idioma
(browser´s locale), inclusive caracteres acentuados em português.

19.2.19-toLocaleUpperCase() - Converte todos os caracteres de um string para maiúsculas, de
acordo com o idioma do browser

19.2.20-toLowerCase() - Converte todos os caracteres do string para letras minúsculas.

19.2.21-toString() - Retorna o valor de um objeto e o converte para string.

19.2.22-toUpperCase() - Converte todos os caracteres de um string para maiúsculas

19.2.23-trim() - Remove os espaços em branco do início e fim do string

Importante: Trim só retira espaços. Quaisquer outro caractere não serão retirados
(como   por exemplo)

19.2.24-valueOf() - Retorna o valor primitivo de um objeto string.
Se é uma string retorna uma string e se é numérico retorna um número.

Importante : A partir deste ponto até o fim dos métodos como String os métodos NÃO são implementados em todos os browsers modernos (IE11 tá fora).

19.2.25-anchor() - Cria uma âncora

19.2.26 - big() - Exibe um string usando uma fonte de letra grande

19.2.27-blink() - Exibe um string piscando
Aparecendo e sumindo - só funciona no Opera
19.2.28 - bould() - Exibe um string em negrito

19.2.29-fixed() - Exibe um string com a fonte monoespaçada (fixed-pitch font)

19.2.30 - fontcolor() - Exibe um string com a cor especificada

19.2.31-fontsize() -Exibe um string utilizando o tamanho de fonte de letra especificado

19.2.32-italics() - Exibe um string utilizando a fonte de letra em itálico

19.2.33-link() - Exibe um string como um hyperlink

19.2.34-small() - Exibe um string usando uma fonte de letra pequena

19.2.35-strike - Exibe uma string riscado no meio (strikethrough)

19.2.36-sub() - Exibe o texto subscrito

19.2.37 - sup() - Exibe o texto superescrito

20-O Objeto Math

Em Javascript, é uma biblioteca de funções matemáticas básicas.
São elas: Arredondamentos, máximo, mínimo, exponenciação, logarítmicos, PI, Raiz Quadrada e Cúbica, Valor Absoluto, Funções trigonométricas entre outras.

Referência: https://www.w3schools.com/jsref/jsref_obj_math.asp

Notas:

1-O JavaScript utiliza algoritmos simples de cálculo o que muitas vezes causa algumas imprecisões. Por exemplo, o resultado era para dar 26 exato e deu 26.00000000000001

2-Sempre que o valor recebido estiver fora do range da função, o resultado retornado será NaN.

3-Sempre que no objeto Math utilizamos log temos a seguinte sintaxe:
LOG(x) é o logaritmo na base 10 de x.
LN(x) é o logaritmo natural (base E) de x.
LOG2(x) é o logaritmo na base 2 de x.

4-Todas as funções trigonométricas utilizam os ângulos em Radianos.

4.1-Tabela dos principais Senos/Cossenos/Tangentes do Círculo Trigonométrico.

4.2-Ângulos e o Círculo Trigonométrico.

5-Para converter o ângulo de radianos para graus basta multiplicar por 180 e dividir por PI.

20.1-Propriedades do objeto Math

20.1.1-E - Retorna o número de Euler

20.1.2-LN2 - Retorna o logaritmo natural de 2 (ln(2))

20.1.3-LN10 -Retorna o logaritmo natural de 10 (ln(10))

20.1.4-LOG2E -Retorna o logaritmo na base 2 do número E(número de Euler)

20.1.5-LOG10E - Retorna o logaritmo na base 10 do número E(número de Euler)

20.1.6-PI - Retorna o valor aproximado de PI

20.1.7-SQRT1_2 - Retorna a raiz quadrada de 1/2

20.1.8-SQRT2 - Retorna a raiz quadrada de 2

20.2-Métodos do objeto Math

20.2.1-abs(x) - Retorna o valor absoluto de x, também chamado de módulo de x

20.2.2-acos(x) - Retorna o arco (ângulo - em radianos) cujo cosseno é o valor recebido pelo parâmetro
Importante: O parâmetro deve estar entre -1 e 1.

20.2.3-acosh(x) - Retorna o arco (ângulo - em radianos) hiperbólico cujo cosseno é o valor recebido pelo parâmetro
É a operação Inversa de Cosseno hiperbólico.
Importante: O parâmetro deve estar entre -1 e 1.

20.2.4-asin(x) - Retorna o arco (ângulo - em radianos) cujo seno é o valor recebido pelo parâmetro
É a operação Inversa de Seno.
Importante: O parâmetro deve estar entre -1 e 1.

20.2.5-asinh(x) - Retorna o arco (ângulo - em radianos) hiperbólico cujo Seno é o valor recebido pelo parâmetro
É a operação Inversa de Seno hiperbólico.

20.2.6-atan(x) - Retorna o arco (ângulo - em radianos) cuja Tangente é o valor recebido pelo parâmetro
É a operação Inversa da Tangente.

Importante: O parâmetro deve estar entre -PI/2 e PI/2.

20.2.7-atan2(y,x) - Retorna o arco tangente do quociente de seus argumentos

Suponha que você tivesse um ponto com as coordenadas (x, y) de (4,8), você poderia calcular o ângulo em radianos entre esse ponto e o eixo X positivo da seguinte forma: atan2(8,4)

20.2.8-atanh(x) - Retorna o arco (ângulo - em radianos) cuja Tangente hiperbólica é o valor recebido pelo parâmetro

É a operação Inversa da Tangente hiperbólica.

Importante1: O parâmetro deve estar entre -PI/2 e PI/2.

Importante1: Não funciona no IE11.

20.2.9-cbrt(x) - Retorna a raiz cúbica de x

20.2.10-ceil(x) - Retorna x, arredondado para cima, até o inteiro mais próximo

Se a parte fracionária for diferente de zero sempre ceil vai arredondar um para cima.

20.2.11-cos(x) - Retorna o cosseno de x (x está em radianos)

20.2.12-cosh(x) - Retorna o cosseno hiperbólico de x

20.2.13-exp(x) - Retorna o valor de E (Número de Euler) elevado a x

20.2.14-floor(x) - Retorna x, arredondado para baixo, para o inteiro mais próximo

A parte fracionária é simplesmente jogada fora, não levada em consideração.

20.2.15-log(x) - Retorna o logaritmo natural (base E) de x

20.2.16-max(x, y, z, ..., n) - Retorna o número com o maior valor

20.2.17-min(x, y, z, ..., n) - Retorna o número com o menor valor

20.2.18-pow(x, y) - Retorna o valor de x elevado a y

20.2.19-random() - Retorna um número aleatório entre 0 e 1

20.2.20-round(x) - Arredonda x para o inteiro mais próximo

Importante: Leva em conta a parte fracionária. Se >= arredonda para cima senão para baixo.

Nota : As funções ceil e floor não levam em conta a parte fracionária do número.

20.2.21-sin(x) - Retorna o seno de x (x está em radianos)

20.2.22-sinh(x) - Retorna o seno hiperbólico de x

20.2.23-sqrt(x) - Retorna a raiz quadrada de x

20.2.24-tan(x) - Retorna a tangente de um ângulo

20.2.25-tanh(x) - Retorna a tangente hiperbólica de um número

20.2.26-trunc(x) - Retorna a parte inteira de um número (x)

21-Manipulando Datas em JavaScript

As funções de manipulação de datas dentro do JavaScript estão encapsuladas dentro do objeto Date.

Portanto, para ter acesso a estas funções a primeira instrução sempre será a instrução para instanciar o objeto Date e pode ser feito da seguinte maneira:

```
var d = new Date();
```

As datas e horários obtidos pelo JavaScript vem de uma consulta ao sistema operacional do micro. Algumas configurações como 'Configurações Regionais' podem mudar o estilo e a formatação da data.

Os métodos setTime, setYear, setDay, setETC...são utilizados para somar ou subtrair horas, dias meses, anos a data. Sendo assim, os métodos set tem a lógica necessária para fazer a função, ou seja, ao somar dias precisa saber se o mês tem tantos dias, se o ano é bissexto, etc..

21.1-Propriedades do Objeto Date

21.1-constructor

Em JavaScript, a propriedade constructor retorna a função construtora para o objeto. Neste caso sempre irá retornar function Date() {[native code] }

21.2-prototype

Permite adicionar novas propriedades a um objeto já existente, tipo, preciso acrescentar o nome do mês em português.

21.2-Métodos do Objeto Date

21.2.1- **getDate** - Retorna o dia do mês da data
21.2.2-**getDay** – Retorna o dia da semana da data
21.2.3-**getFullYear** – Retorna o ano da data com 4 dígitos
21.2.4-**getHours** – Retorna as horas da data
21.2.5-**getMilliseconds** – Retorna os milissegundos da data
21.2.6-**getMinutes** – Retorna os minutos da data
21.2.7-**getMonth** – Retorna o mês da data
21.2.8-**getSeconds** – Retorna os segundos da data
21.2.9-**getTime** - Retorna o número de milissegundos desde meia-noite de 1º de janeiro de 1970 e uma data especificada
21.2.10-**getTimezoneOffset** - Retorna a diferença de horário entre a hora UTC e a hora local, em minutos
21.2.11-**getUTCDate** - Retorna o dia do mês, de acordo com o horário universal
21.2.12-**getUTCDay** - Retorna o dia da semana, de acordo com o horário universal
21.2.13-**getUTCFullYear** - Retorna o ano, de acordo com o horário universal
21.2.14-**getUTCHours** - Descrição: Retorna a hora, de acordo com o horário universal
21.2.15-**getUTCMilliseconds** - Retorna os milissegundos, de acordo com o tempo universal
21.2.16-**getUTCMinutes** - Retorna os minutos, de acordo com o horário universal
21.2.17-**getUTCMonth** - Retorna o mês, de acordo com o horário universal
21.2.18-**getUTCSeconds** - Retorna os segundos, de acordo com o tempo universal
21.2.19-**getYear** - Descontinuada. Use o método getFullYear
21.2.20-**now** - Retorna o número de milissegundos desde a meia-noite de 1º de janeiro de 1970

21.2.21-**parse** - Analisa uma string de data e retorna o número de milissegundos desde 1º de janeiro de 1970.
21.2.22-**setDate** -Define o dia do mês de um objeto de data – Utilizada para definir uma data qualquer.
21.2.23- **setFullYear** - Define o ano de um objeto de data – Utilizado para somar anos a data.
21.2.24- **setHours** - Define a hora de um objeto de data – Utilizado para somar horas a data.
21.2.25- **setMilliseconds** - Descrição: Define os milissegundos de um objeto de data – Utilizado para somar milisegundos a data.
21.2.26- **setMinutes**- Define os minutos de um objeto de data – Utilizado para somar minutos a data.
21.2.27-**setMonth** - Descrição: Define o mês de um objeto de data – Utilizado para somar meses a data.
21.2.28-**setSeconds** - Define os segundos de um objeto de data. – Utilizado para somar segundos a data.
21.2.29-**setTime**-Define uma data para um número especificado de milissegundos após / antes de 1º de janeiro de 1970
21.2.30-**setUTCDate** - Define o dia do mês de um objeto de data, de acordo com o horário universal
21.2.31-**setUTCFullYear** - Define o ano de um objeto de data, de acordo com o horário universal – Utilizado para somar anos a data.
21.2.32-**setUTCHours** - Descrição: Define a hora de um objeto de data, de acordo com a hora universal – Utilizado para somar horas a data.
21.2.33-**setUTCMilliseconds** - Define os milissegundos de um objeto de data, de acordo com a hora universal – Utilizado para somar milissegundos a data.
21.2.34-**setUTCMinutes** - Defina os minutos de um objeto de data, de acordo com a hora universal – Utilizado para somar minutos a data.
21.2.35-**setUTCMonth**- Define o mês de um objeto de data, de acordo com o horário universal – Utilizado para somar meses a data.
21.2.36-**setUTCSeconds** - Definir os segundos de um objeto de data, de acordo com a hora universal – Utilizado para somar segundos a data.
21.2.37-**setYear**-Descontinuada. Use o método setFullYear ()
21.2.38-**toGMTString** - Descontinuada. Use o método toUTCString()
21.2.39-**toJSON** - Retorna a data como uma string, formatada como uma data JSON
21.2.40-**toLocaleDateString** - Retorna a parte da data de um objeto Date como uma string, usando convenções de localidade.
21.2.41-**toLocaleTimeString** - Retorna a parte de hora de um objeto Date como uma string, usando convenções de localidade
21.2.42-**toLocaleString** - Converte um objeto Date em uma string, usando convenções de localidade
21.2.43-**toString** - Converte um objeto Date em uma string
21.2.44-**toTimeString** - Converte a parte de hora de um objeto Date em uma string
21.2.45-**toUTCString** - Converte um objeto Date em uma string, de acordo com o horário universal
21.2.46-**UTC** - Retorna o número de milissegundos em uma data desde a meia-noite de 1º de janeiro de 1970, de acordo com a hora UTC
21.2.47-**valueOf** - Retorna o valor primitivo de um objeto Date

21.3-Operações com data

Como foi dito no item 21, os métodos setxxx foram feitos para fazer as operações com datas e horários. Por exemplo, ao somar 5 horas num horário como 22hs30min50seg o objeto

Date do Javascript é suficientemente inteligente para saber que um dia tem 24 horas e fará a operação corretamente desprezando o que sobrar das 24 hs e soma 1 ao dia seguinte da data.

Como exemplo, cito uma funcionalidade para somar dias a uma data:

```
<script>
   var d = new Date();
   document.write("A data atual é : ")
   document.write(d.toString());
   document.write("<br>")

   document.write("daqui a 30 dias será : ")
   d.setDate(d.getDate() + 30);          //<---SetDate
   document.write(d.toString());
</script>
```

22-Eventos do mouse

Referência : https://www.w3schools.com/tags/ref_eventattributes.asp

Podemos fazer que certas funcionalidades sejam ativadas por eventos relacionados ao mouse.

Praticamente todos os elementos html podem ter eventos associados a eles.

Os eventos do mouse são os seguintes:

22.1-**onclick** – Disparado quando o elemento é clicado.
22-2-**ondblclick** - Disparado quando o elemento sofre um click duplo.
22.3-**onmousedown** - Disparado quando algum botão do mouse é pressionado.
22.4-**onmousemove** - Disparado quando o mouse é movido.
22.5-**onmouseout** - Disparado quando o elemento deixa a área do elemento.
22.6-**onmouseover** - Disparado quando o mouse passa sobre o elemento.
22.7-**onmouseup** – Disparado quando o botão do mouse é liberado.
22.8-**onmousewheel** - Descontinuada. Usar onwheel em seu lugar.
22.9-**onwheel** - Disparado quando o mouse está sobre o elemento e acionamos a rodinha de scroll do mouse.

Ex. de uso : < div onclick=”função1” onmouseout=”funcao2”>Oi

23.Eventos de teclado

23.1-onkeydown

Disparado quando uma tecla é pressionada

Este evento permite interceptar uma tecla antes que ela seja enviada para o browser.

23.2-onkeyup

Disparado quando uma tecla é liberada

Este evento NÃO permite interceptar uma tecla antes que ela seja enviada para o browser.

23.3-onkeypress

Disparado quando eu mantenho uma tecla pressionada.

Ocorre entre onkeydown e onkeyup

Este evento só é disparado se a tecla representa um caractere. Teclas como control, shift, alt não disparam este evento

24.Eventos de janela

24.1-onresize

Ocorre quando redimensionamos a janela do browser.

Este evento é exclusivo da tag body.

24.2-onscroll

Ocorre quando giramos a rodinha do mouse e existe a barra lateral para deslocamento vertical da página.

25. Eventos de formulário

25.1-onfocus

Este evento é disparado quando o formulário recebe o foco do cursor do mouse.

25.2-onblur

Este evento é disparado quando o formulário perde o foco do cursor do mouse.

25.3-onchange

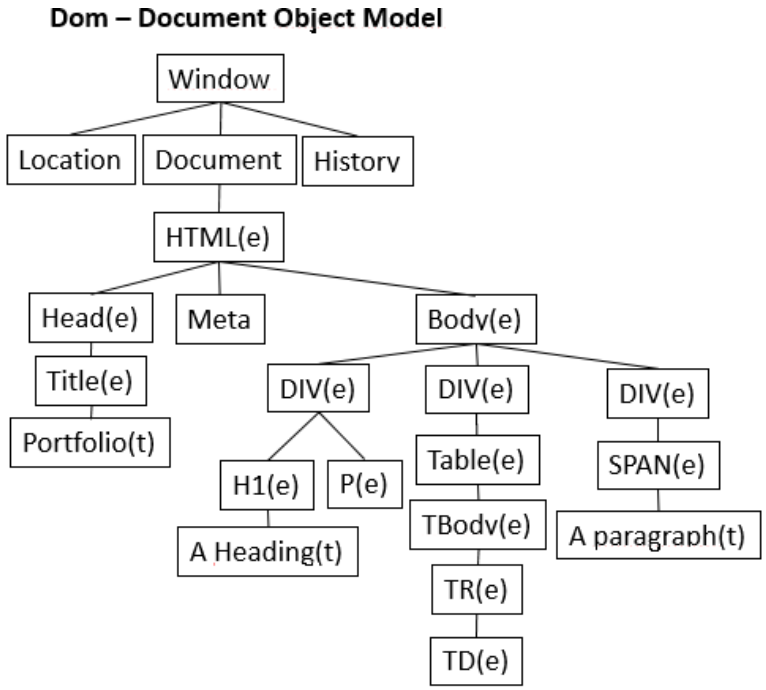
Este evento é disparado quando o estado do elemento é modificado.

Muito usado no elemento select do html.

25-DOM

Referência : https://www.w3schools.com/js/js_htmldom_elements.asp

É como o javascript acessa os elementos do documento html, dentro de um modelo de estrutura hierárquica como:



Legenda : e=Element, t=Text, a=Attribute

21. Acessando elementos do documento HTML via DOM

Podemos selecionar um element do documento html por :

1-id
2-nome da tag (tag name)
3-nome da classe (class name)
4-CSS selectors
5-Coleção dos objetos do Documento HTML (HTML object collections)

21.1-ID

Seleciona o elemento pelo ID. Note que apenas um elemento do documento é selecionado já que o ID de um elemento dentro do HTML deve ser único.
Ex.: var myElement = document.getElementById("intro");

21.2- nome da tag (tag name)

Devolve uma coleção de elementos com todas as tags da classe selecionada.
Ex.: var x = document.getElementsByTagName("p"); //retorna todas as tags p do documento.

22-Arrays

Arrays são listas de itens associados a uma única variável vetorial. Por exemplo, se eu quisesse armazenar o nome das frutas de uma salada de frutas, cada nome de fruta utilizaria uma variável. Criando um Array SaladaFrutas (que seria uma lista) armazenaríamos todas as frutas numa única variável.

A principal vantagem do elemento array é que fica mais fácil trabalhar com itens agrupados no array.

Exemplo: Se eu quisesse saber se na lista de frutas tem uva ... eu teria que fazer if(fruta1=’uva’)... if(fruta2=’uva’) ... if(fruta3=’uva’) .... . Com o array não é necessário, a gente coloca tudo no array e através do índice ou chave a gente seleciona um elemento da lista.

22.1-Declarando o array

Método 1: var listaDeFrutas = Array() //parenteses

Método 2: var listaDeFrutas = Array('banana', 'maçã','morango','uva') //entre parenteses
Nota: Neste método o índice inicial é 0, o segundo 1 e assim por diante. Sempre numérico.

Método 3: var listaDeFrutas = [] //[] é um alias para Array()

Método 4: var listaDeFrutas = ['banana', 'maçã','morango','uva'] //entre colchetes
Nota: Neste método o índice inicial é 0, o segundo 1 e assim por diante. Sempre numérico.

22.2-Populando os dados do array

Método 1:

```
ListaDeFrutas[0]='banana';
ListaDeFrutas[1]='maçã';
ListaDeFrutas[2]='morango';
ListaDeFrutas[3]='uva';
ListaDeFrutas['x']='carambola';
```

Nota: O índice de um array quando designado por nós pode ser numérico ou alfabético.

Nota: Os métodos de inclusão e exclusão dos elementos de um array podem atuar no inicio ou no final da pilha.

22.3-Incluindo elementos num array existente

22.3.1-Incluindo no final do array

Sintaxe: NomeArray.push('novo elemento')

22.3.2-Incluindo no inicio do array

Sintaxe: NomeArray.unshift('novo elemento')

22.4-Removendo elementos num array existente

22.4.1-Excluindo do final do array

NomeArray.pop()

22.4.2-Excluindo do início do array

NomeArray.shift()

22.5-Arrays – Métodos de pesquisa

Existe um método para procurar pela ocorrência de um valor dentro do array. É o método indexOf()

22.6-Arrays -Ordenação

Caso seja necessário ordenar arrays existe a função sort para isto.

22.6.1-Arrays – Ordenação – Strings

NomeArray.sort()

22.6.2- Arrays – Ordenação –Numérico

É o mesmo método NomeArray.sort() mas como a ordenação é feita em ASCII é preciso que uma função indique como fazer a ordenação corretamente:

```
<script>
  //a-b
  //se < 0 : a deve ser colocado antes de b
  //se > 0 : b deve ser colocado antes de a
  //se = 0 : não precisa ordenar, já esta ordenado
  function ordernarNumeros(a,b){
    return a-b;
  }
</script>
  ListaNum.sort(ordernarNumeros);
```

32-Estruturas de repetição

As estruturas de repetição possibilitam executar um bloco de código uma ou mais vezes dependentes de fatores e condições.

32.1-While

Uso: while(condição){}

Funcionamento: caso a condição seja verdadeira o bloco de código dentro das chaves será executado. Portanto, primeiro testa e depois executa.

32.1.1-break

Permite sair do loop while. A execução passa a ser feita a partir do fim do bloco de código do while({}).

32.1.2-continue

Dentro do bloco de código while podemos ter situações em que desejamos que o restante do bloco não seja executado, mas sim retorne ao começo do bloco de código while. Para este fim existe o continue.

Cuidado com esta instrução porque é muito fácil ficar num loop infinito se a instrução de incremento ficar depois do continue.

32.2-Do While

Uso : do {} while (condição)

Esta estrutura de repetição difere do while porque pelo menos uma vez o bloco é executado antes que a condição seja testada.

32.3-for

Uso: for (inicialização;condição;incremento/decremento){} ou

for (variável;condição;incremento/controle){}

A primeira parte do comando for é a inicialização que define o ponto de partida que será utilizado posteriormente pela condição.

A segunda parte é a condição que é o que permite sair do loop for.

A terceira parte é uma operação que muda o valor de inicialização e posteriormente permitirá a saída do loop pela condição.

32.4- for in

Esta instrução é específica para varrer os elementos de uma coleção.

Uso: for(var x in Lista){}

Vantagens de utilizar esta instrução: Quando não sabemos exatamente o conteúdo da lista e desejamos saber e varrer apenas os itens existentes.

Nota: Caso os índices da Lista sejam totalmente aleatórios tanto no valor como no tipo esta instrução é muito conveniente para varrer os itens quando não se tem a mínima ideia dos índices da lista.

32.5-for each

Segue o mesmo principio do 'for in' que é uma instrução específica para varrer os elementos de uma coleção, mas com a diferença que ela retorna os valores do item, o índice do item e o array inteiro como parâmetros de saída.

Ex. de Uso:

```
<script>
  var ListaFuncionarios = ['Viviane', 'Rosângela', 'Miguel', 'Lucas', 'Fernanda'];

  ListaFuncionarios.forEach(
    function (valor, indice,array){
      document.write(valor);
      document.write("<br>");
      document.write(indice);
      document.write("<br>");
      document.write(array);
      document.write("<br>----------------------------------------------------------------------------<br>");
    }
  )
</script>
```

Importante: o element for each reconhece apenas os índices numéricos de 0 a n. Portanto se na matriz tiver índices alfabéticos, lógicos, números negativos eles serão ignorados.

Ex.2 :

```
<script>
  var ListaFuncionarios = ['Viviane', 'Rosângela', 'Miguel', 'Lucas', 'Fernanda'];

  var f = function(valor, indice, array){
      document.write(valor);
      document.write("<br>");
      document.write(indice);
      document.write("<br>");
      document.write(array);
      document.write("<br>----------------------------------------------------------------------------<br>");
  }
```

```
    ListaFuncionarios.forEach(f);
  </script>
```

33-Parâmetros de uma função

Sempre que uma função é chamada o conjunto de parâmetros que ela recebe vem em um vetor com o nome arguments. Com isto podemos montar funções super flexíveis quanto aos parâmetros, tanto em número quanto em tipo. Veja o exemplo:

```
  <script>
    function x(){
      for (var i in arguments){
        document.write("Parametro : Indice:");
        document.write(i);
        document.write(", Valor :");
        document.write(arguments[i]);
        document.write("<br>");
      }
    }


    //Pode-se chamar a função x com qualquer quantidade e tipo de parâmetro

    document.write("Parametros Numéricos : <br>");
    x(1,2,3,4,5,6,7,8)
    document.write("<br>");
    document.write("Parametros String : <br>");
    x('P1','P2','P3')
    document.write("<br>");
    document.write("Parametros Lógicos : <br>");
    x(true,false,1>3,7>1)      //exibe Parametro : 0, Valor :true      Parametro : 1, Valor :false
                                  //Parametro : 2, Valor :false     Parametro : 3, Valor :true

    document.write("<br>");
    document.write("Parametros Matrizes : <br>");
    var y = Array()
    y[0]='pimentão'
    y[1]='quiabo'
    x(y);         //exibe: Parametro : 0, Valor :pimentão,quiabo
  </script>
```

34-BOM – Browser Object Model

Referência:

É uma maneira do JavaScript se comunicar com recursos do browser como acessar o histórico de navegação, redirecionamento de urls, salvar e recuperar cookies.

Este recursos não estão ligados aos elementos HTML mas sim a ações disparadas pelos recursos do próprio browser.

Através da API DOM (Document Object Model) podemos acessar os elementos HTML da página e através do BOM (Browser Object Model) podemos acessar os recursos do browser.

O BOM é composto pelos componentes:

window	- manipula os elementos do DOM
screen	- manipula os elementos de janela
location	- manipula os elementos de navegação
history	- manipula os elementos de histórico de navegação
navigator	- manipula os elementos internos do navegados
Alert(PopUp)	- PopUp de alerta para o usuário
Timing	- Permite criar timers (como para reload da página).
cookies	- Armazena ou recupera informações

34.1- window

Representa a janela do Browser onde elementos do documento são renderizados.

Portanto, temos a hierárquia : window.document.

Todas os objetos globais do JavaScript são propriedades deste objeto e por este fato todas as variáveis automaticamente tornam-se membros deste objeto.

Variáveis globais são propriedades deste objeto.

Funções globais são métodos deste objeto.

Até o DOM é uma propriedade deste objeto.

Exemplos de utilização:

34.1.1 - window.innerHeight

Retorna a altura da janela do browser sem contar as barras de rolagem.

34.1.2-window.innerHeight

Retorna a largura da janela do browser sem contar com as barras de rolagem.

34.1.3-window.open()

Abre uma nova janela

34.1.4-window.close()

Fecha uma janela

34.1.5-window.moveTo()

Move a janela corrente

34.1.6-window.resizeTo()

Redimensiona a janela corrente

34.2-screen

Manipula os elementos de janela

34.3-location

Manipula os elementos de navegação

34.4-history

Manipula os elementos de histórico de navegação

34.5-navigator

manipula os elementos internos do navegados

34.6-Alert(PopUp)

PopUp de alerta para o usuário

34.7-Timing

Permite criar timers (como para reload da página).

34.8-cookies

Armazena ou recupera informações na máquina do cliente

35-JavaScript e a Orientação a objetos

35.1- Objeto

É uma entidade do mundo real que queremos representar no mundo virtual de programação. Chamamos isto de representação do mundo real.

Por exemplo, se você está fazendo uma aplicação para uma agência de automóveis provavelmente vai existir uma entidade atribuída a finalidade da maioria das agências de automóveis, que é vender carros. Portanto, carro seria uma entidade óbvia.

Outro exemplo, se você está fazendo uma aplicação para um banco provavelmente vai existir uma entidade atribuída a finalidade da maioria dos bancos, que é conta bancária.

35.2-Pilares das linguagens orientadas a objetos:

2.1-Abstração
2.2-Encapsulamento
2.3- Herança
2.4- Polimorfismo

35.2.1-Abstração

É a capacidade de representarmos as características, modelos, dados de um objeto no mundo virtual de programação.

Por exemplo, se o objeto é um carro o que precisamos saber sobre o carro para atender as necessidades da aplicação. Por exemplo, cor, preço, ano de fabricação, estado, último dono, etc... De acordo com a necessidade da aplicação escolheremos o que iremos representar.

É um pilar puramente intuitivo e no levantamento de requisito das aplicações podemos levantar quais particularidades do elemento precisaremos abstrair.

A abstração é composta por:

2.1.1-Entidade
2.1.2-Identidade
2.1.3-Características
2.1.4-Ações / Métodos

35.2.1.1-Entidade

É o objeto do mundo real abstraído no nosso mundo virtual de programação.
Ex.: Carro, Conta bancária

Normalmente este item é apenas uma representação do elemento virtual e não o elemento virtual em si. Por exemplo, a receita de bolo é uma entidade enquanto o 'bolo' (virtual) seria o objeto.

35.2.1.2-Identidade

É uma representação da Entidade no nosso mundo virtual.

Diferentemente da Entidade, a Identidade é o elemento representado no nosso mundo virtual, ou seja, seria nosso 'bolo' virtual.

Explicando melhor, se numa aplicação de agência de automóveis temos a entidade carro, a Identidade 'carro' representaria cada carro que a agência negocia.

35.2.1.3-Características

São as particularidades de cada Identidade. Cada carro teria sua placa, ano, modelo, cor.

35.2.1.4-Ações / Métodos

São as capacidades ou o que se pode fazer com as Identidades. Num carro, por exemplo, podemos acelerar, frear, parar.

Exemplificando melhor:

Entidade	Identidade	Características	Ações
Carro	X = new Carro()	Marca, modelo, cor, quantidade de portas	Ligar, acelerar, frear, desligar
Sonho	Y = new Sonho()	Tipo, história, gostaria_repetir_sonho	exibirHistoria
Conta_Bancária	Z= new Conta_Bancária ()	Agência, NúmeroConta, Saldo, Limite	depositar, sacar, consultarSaldo

35.2.2-Encapsulamento

Consiste em uma camada de segurança onde toda a lógica para trabalhar com o elemento esteja embutida e isolada de processos externos. Com isto garantimos integridade e confiabilidade na aplicação das regras necessárias ao tratamento dos elementos.

Os objetivos do encapsulamento são:

2.2.1-Segurança

2.2.2-Reutilizável

Com o encapsulamento o programador não precisa se preocupar com detalhes internos dos processos, basta utilizar os métodos que eles garantem a perfeita execução da tarefa abstraindo a complexidade envolvida no processo.

A única desvantagem apresentada por este pilar é a necessidade de gerar mais código para ser implementado. Por exemplo, se você precisa ler ou alterar uma variável como ela pertence ao objeto você deverá fazê-lo pelos métodos get e set dentro do próprio objeto, coisa que sem este pilar faríamos acessando diretamente a variável. Como sempre, poder é inimigo da segurança.

35.2.3- Herança

Podemos permitir que uma classe herde as características de uma classe pai através do operando extends.

Ex.: class papagaio extends ave(){}

Neste caso todas as propriedades, métodos da classe ave é passada para a classe papagaio.

35.2.4- Polimorfismo

Polimorfismo é a propriedade que uma classe ser escrita de diversas formas gerando 'assinaturas' diferentes e com características diferentes cada vez que é escrita ou herdada ou reescrita.

Por exemplo: classe ave(){} (1) class ave(penagem){} (2)

Ao instanciar ou chamar a classe: ave(verde) irá chamar a classe 2

ave() irá chamar a classe 1

var x as new ave() (1)

var x as new ave('azul') (2)