Características da linguagem

- Interpretada;
- Confinada no navegador;
- Baseada em objetos;
- Residente na máquina cliente;
- Dirigida a eventos;
- Código-fonte embutido no próprio HTML;
- Independente de plataforma;
- Fracamente tipada;
- Sensível ao caso (variáveis, funções e operadores);
- Última versão do núcleo da linguagem: ECMAScript 2021;
- Veja mais sobre versões em http://en.wikipedia.org/wiki/JavaScript#Versions

Inserir JavaScript

- Através da tag <script>;
- Em qualquer parte do código HTML;
- Preferencialmente, ao final do documento, antes do fechamento de </body>.

```
<html>
<head>
<title>A Minha Página com JavaScript</title>
<script>
   alert("Seja bem vindo(a) à minha página!");
</script>
</head>

<body>
   Aqui colocamos o conteúdo da página em HTML
<script>
   var valida = true;
</script>
</body>
</html>
```

JavaScript dentro da seção <body>

```
<html>
<head>
<title>A Minha Página com JavaScript</title>
</head>
<body>
   <!-- Elementos em HTML -->
   <script>
    alert("JavaScript dentro do corpo do documento HTML!");
   </script>

</body>
</html>
```

Inserir JavaScript

- Também pode ser inserido através da chamada a um arquivo externo, com a extensão .js:

```
<html>
<head>
<title>A Minha Página com JavaScript</title>
<script src="arquivo.js"></script>
</head>
<body>
   Aqui colocamos o conteúdo da página em HTML
</body>
</html>
```

Diferenças na localização da chamada

- JavaScript na seção <head>: todo o código JavaScript nesta área é executado ***ANTES*** de qualquer coisa, mesmo antes de o navegador renderizar o elementos HTML no corpo da página – exceção feita somente para funções;
- JavaScript na seção <body>: o código é executado quando a tag <script> for encontrada. Neste caso, podemos ter JavaScript sendo executado ***antes ou depois*** de o navegador renderizar determinados elementos;
- JavaScript externo: funciona como se a chamada ao arquivo externo fosse substituída pelos comandos constantes no arquivo. Dependendo de onde a chamada ao arquivo externo foi colocada, o código será executado antes (se chamado na seção <head>) ou antes ou depois (seção <body>).

Comentários em JavaScript

- Comentário de uma única linha:

```
//comentário
```

- Comentário de várias linhas:

```
/* comentário .....
de múltiplas.....
.....linhas.....*/
```

Variáveis em JavaScript

- Toda variável em JavaScript:
  - ✓ Deve começar por uma letra ou o caracter subinhado (_) ou $;
  - ✓ A partir do segundo caracter, pode haver uma combinação, em qualquer ordem, de letras, números, sublinhado ou cifrão;
  - ✓ Deve ser precedida pela declaração **let**;
  - ✓ Variáveis declaradas com let têm escopo e são reconhecidas somente dentro do bloco ou expressão (lógica, numérica, etc.) em que foram criadas;
  - ✓ Variáveis criadas com let não podem ser redeclaradas.
- JavaScript **diferencia** maiúsculas de minúsculas em nomes de variáveis e funções;
- Pode-se criar uma variável, também, por meio da partícula **var** ou, até mesmo, usando diretamente o nome da variável, sem nenhuma declaração antes. Evite estes métodos, pois trazem problemas quando usados em códigos e estão disponíveis, ainda, apenas para efeito de retrocompatibilidade com códigos antigos.

Tipos de dados

- São seis os tipos de dados do JavaScript:
  - ✓ **Numérico** (inteiro ou real);
  - ✓ **Booleano** (verdadeiro ou falso);
  - ✓ **String** (texto – qualquer sequência de caracteres entre aspas " ou apóstrofo ');
  - ✓ **Objeto (aqui incluídos funções e vetores – arrays)**;
  - ✓ **Symbol** (usado como um identificador único dentro do código – útil para a criação automática de nomes de propriedades relacionadas a objetos);
  - ✓ **Null** (nulo) – tipo especial de dado, significando um valor nulo – ponteiro de objeto vazio, ausência de objeto;
  - ✓ **Undefined** (indefinido) - ausência de valor – variáveis não inicializadas, parâmetros de funções faltando, referência a propriedades inexistentes de um objeto.

Conversão de valores - I

- **parseInt**(string) – converte um conjunto de caracteres numéricos em um inteiro;
- **parseFloat**(string) – converte um conjunto de caracteres numéricos (com um possível ponto decimal) para um valor real;
- Outra forma de converter uma sttring numéricaa é por meio da função Number().;
- Exemplo: var idade = "67";
- Após invocar Number(idade), a string "67"será tranformada no número 67;
- Se, em qualquer caso, a conversão não puder ser feita, tem-se a mensagem de erro com o código **NaN** – Not a Number (Não é um Número);

Conversão de valores - II

- Em JavaScript, qualquer tipo de dado pode ser convertido em um valor booleano de acordo com a tabela abaixo:

Tipo de dado	Valores convertidos para **true**	Valores convertidos para **false**
Boolean	`true`	`false`
String	Any nonempty string	`""` (empty string)
Number	Any nonzero number (including infinity)	`0`, `NaN` (See the "NaN" section later in this chapter.)
Object	Any object	`null`
Undefined	n/a	`undefined`

Teste de valores numéricos vindo de formulários

- Usamos um artífico com as funções parseInt() e parseFloat(), vistas anteriormente. Veja o exemplo abaixo:

```
<html> <head> <script>
   function testar(valor)
      {
      if(parseInt(valor) == valor || parseFloat(valor) == valor)
         alert("É um número");
      else
         alert("Não é um número!");
      }
</script>
</head> <body>
   <form>
      <label> Nome: </label>
      <input type="text" name="nome" onblur="testar(this.value);">
   </form>
</body> </html>
```

Expressões literais

- Valores (numéricos, texto, booleano ou nulo) declarados textualmente e associados a alguma variável dentro do script – no exemplo, valores destacados em vermelho são expressões literais;
- Exemplo:

```
<script>
   var nome = "visitante";
   var hora = 11;
   if(hora < 12)
      document.write("Bom dia. Seja bem vindo senhor(a) " + nome);
   else
    {
      if(hora >= 13)
         document.write("Boa tarde. Seja bem vindo senhor(a) " +
   nome);
      else
         document.write("Seja bem vindo! Almoça conosco?");
    }
</script>
```

Números inteiros

- O JavaScript pode trabalhar com:
  - ✓ Números na base 10 – sequência de dígitos que nunca começa com zero;
  - ✓ Números na base 8 – sequência de dígitos que começam com 0;
  - ✓ Números na base 16 – sequência de dígitos que começam com 0X ou 0x;
- Exemplo:

```
<script>
 var i = 42; // decimal
 var j = 052; // octal
 var k = 0X2A (ou 0x2a); // hexadecimal
 // quando executar este código repare que as variáveis  têm todas
o mesmo valor
     document.write("i = " + i + "<br>");
     document.write("j = " + j + "<br>");
     document.write("k = " + k);
</script>
```

- Inteiros octais ou hexadecimais são automaticamente convertidos para decimais em qualquer operação aritmética.

Expressões de texto (strings)

- São representadas entre aspas (" ") ou apóstrofos (' ');
- Podem conter os caracteres especiais da tabela abaixo, chamados caracteres de formatação:

Caractere	Significado
\b	backspace
\f	form feed
\n	new line
\r	carriage return
\t	tab
\\	backslash

- A barra invertida (\) funciona como caracter de escape para escrever aspas e apóstrofos dentro da expressão literal. Exemplo:

```
<script>
  var texto = "Ele leu o \"Auto da Barca do Inferno\" de Gil
  Vicente.";
   document.write(texto);
</script>
```

Operadores de atribuição

- O operador de atribuição em JavaScript é o sinal de igualdade (=). Porém, a linguagem permite a mescla deste operador com outros operadores, como, por exemplo, os aritméticos. Veja:

Versão curta	Significado
x+=y	x=x+y
x-=y	x=x-y
x*=y	x=x*y
x/=y	x=x/y

- Notar que o símbolo + também funciona como operador para concatenação de texto. Por isso, sempre que, numa expressão, você misturar números estrings, ao somar estas variáveis, o JavaScript converte TUDO automaticamente para string e faz a CONCATENAÇÃO.

Operadores relacionais

- Permitem a comparação de valores e sempre retornam um valor booleano (true ou false). Veja a tabela a seguir:

Operador	Descrição	Exemplo
Igualdade (==)	Verifica se os dois operandos são iguais	x==y dá true se x igualar y
Desigualdade (!=)	Verifica se os operandos são desiguais	x!=y dá true se x não for igual a y
Maior do que (>)	Verifica se o operando da esquerda é maior do que o da direita	x>y dá true se x for maior do que y
Maior ou igual (>=)	Verifica se o operando da esquerda é maior ou igual ao da direita	x>=y dá true se x for maior ou igual a y
Menor do que (<)	Verifica se o operando da esquerda é menor do que o da direita	x<y dá true se x for menor do que y
Menor ou igual (<=)	verifica se o operando da esquerda é menor ou igual ao da direita	x<=y dá true se x for menor ou igual a y

- Além destes, temos os operadores de igualdade absoluta (===) e de desigualdade absoluta (!==). Estes dois operadores forçam a comparação dos tipos de dados. Portanto, "0" sempre será diferente de 0 com este operador, isto é, "0" === 0 é false.

Operadores aritméticos

Operador	Descrição	Exemplo(s)
+	Soma valores.	`a = 2 + 3;` `b = b +1;`
-	Subtrai valores (como operador binário).	`x = x - 5;` `x = a - b`
-	Muda sinal (como operador unitário).	`x = -x;` `x = -(a + b);`
*	Multiplica valores.	`a = 2 * 3;` `b = c * 5;`
/	Divide valores.	`a = 50 / 3;` `b = b * 4;`
%	Resto da divisão.	`d = 5 % 3;` d assume valor 2.
++(var)	Incremento de 1 (antes).	Se x é 2, `y = ++x` faz x igual a 3 e depois y igual a 3.
(var)++	Incremento de 1 (depois).	Se x é 2, `y = x++` faz y igual a 2 e depois x igual a 3.
--(var)	Decremento de 1 (antes).	Se x é 2, `y = --x` faz x igual a 1 e depois y igual a 1.
(var)--	Decremento de 1 (depois).	Se x é 2, `y = x--` faz y igual a 2 e depois x igual a 1.

Operadores lógicos

- Retornam um valor verdadeiro ou falso.

Operador	Descrição	Exemplo(s), supondo a = 3 e b = 5
&&	E lógico: retorna verdadeiro se ambas as expressões são verdadeiras e falso nos demais casos	`a==3 && b<10` // retorna verdadeiro `a!=3 && b==5` // retorna falso
||	OU lógico: retorna verdadeiro se pelo menos uma das expressões é verdadeira e falso se todas são falsas	`a==3 || b<10` // retorna verdadeiro `a!=3 || b==5` // retorna verdadeiro `a==1 || b==3` // retorna falso
!	NÃO lógico: retorna verdadeiro se o operando é falso e vice-versa	`! (a==3)` // retorna falso `! (a!=3)` // retorna verdadeiro

Operadores de strings

Operador	Descrição	Exemplo(s)
+	Concatenar strings.	str_1 = "Bom"; `str_2 = str_1 + " dia";` str_2 contém "Bom dia"

+=	Concatenar e atribuir o resultado ao operando da esquerda.	`str_1 = "Bom";`
		`str_1 += " dia";`
		
		str_1 contém "Bom dia"

Outros operadores

Operador	Descrição	Exemplo(s)

?:	Operador condicional na forma:
	condição? exp_1 : exp_2
	Se condição é verdadeira, retorna exp_1 e, se é falsa, retorna exp_2.
		`val = com_frete? "110,00" : "100,00";`
		`document.write("Preço: " + val);`
		
		Se *com_frete* é verdadeiro, escreve:
		"Preço: 110,00" Senão, escreve "Preço: 100,00"

,	Operador vírgula na forma: exp_1, exp_2 Calcula ou avalia ambas as expressões.	Uso comum é em loops tipo for: `for(m=0,n=0; m<5; m++,n++){...`

delete	Exclui um objeto, uma propriedade de objeto, um elemento de uma array ou uma variável explicitamente declarada com `var`. Para elemento de array, não altera o seu tamanho. Apenas o elemento fica indefinido.	`val = new Array(10);`
		`delete val[4];`
		`delete val;`
		`var nivel = 20;`
		`delete nivel;`

new	Cria um novo objeto entre aqueles já existentes na linguagem (str = new String("abc"), etc) ou cria um objeto definido pelo usuário.	`function prod(nome,unidade,valor){`
		`this.nome = nome;`
		`this.unidade = unidade;`
		`this.valor = valor;`
		`}`
		`P01 = new prod("banana","kg",2.00);`

this	Faz referência ao objeto em uso.	`<b>Digite 6 a 10 caracteres</b>`
		`<input type="text" name="ab" size=10`
		`onChange="verificar(this)">`
		
		A função *verificar* (a definir) recebe como parâmetro a caixa de texto, o que permite verificar a quantidade de caracteres digitados.

typeof	Retorna uma string indicativa do tipo de operando.	`var val = 10;`
		typeof val retorna "number"
		
		`str = "abc";`
		typeof str retorna "string"

void	Indica uma expressão para ser avaliada mas sem retornar nenhum valor.	O código abaixo cria um hyperlink nulo, isto é, nada abre: `<a href="javascript:void(0)">Este link nada abre</a>`

Atribuição de valores de variáveis

- A atribuição de valores a variáveis pode ser feita na mesma linha.

```
<script>
  var idade = 25, peso = 72, nome = "Antônio da
  Silva";
</script>
```

- Várias variáveis podem ser inicializadas com um mesmo valor na mesma linha também:

```
<script>
  var idade = peso = diasFimAno = 75;
</script>
```

Estruturas de seleção – if

- Testa uma única possibilidade. Exemplo:

```
<script>
    var resposta = prompt("Qual seu nome?");
    if(resposta == "Pedro")
        {
        alert("Olá, Pedro, tudo bem?");
        resposta = null;
        }
</script>
```

- Se a expressão lógica (resposta == "Pedro) resultar verdadeira, os comandos dentro do if serão executados. Caso contrário, não. Se houver mais de uma linha de comando, as chaves são necessárias.

If...else

- Testa duas possibilidades. Se a expressão lógica dentro do if for verdadeira, os comandos dentro do if são executados. Caso contrário, os comandos dentro do else serão executados. Exemplo:

```
<script>
   var resposta = prompt("Qual seu nome?");
   if(resposta == "Pedro")
    alert("Olá, Pedro, tudo bem?");
   else
    alert("Você não é o Pedro!");
</script>
```

- Novamente, se houver mais de uma linha de comando dentro do if e/ou dentro do else, o uso de chaves { } faz-se necessário.

If...else...if

- Quando temos mais de duas possibilidades a serem testadas, podemos usar a construção mostrada no exemplo a seguir:

```
<script>
    var resposta = prompt("Digite um número inteiro:");
    if(resposta > 0 && resposta < 10)
        alert("Você digitou um valor entre 0 e 10");
    else if(resposta >= 10)
        alert("Você digitou 10 ou um valor maior que 10");
    else if(resposta == 0)
        alert("Você digitou o valor 0");
    else
        alert("Você digitou um valor negativo");
</script>
```

A instrução switch

- No exemplo abaixo, a estrutura testa se o número digitado está no conjunto {6, 7, 10} ou no conjunto {20, 21}. Caso não esteja em nenhum deles, o default é executado.

```
<script>
    var num = prompt("Digite um número inteiro:");
    num = parseInt(num);//transforma string em inteiro
    switch(num)
            {
            case 6:
            case 7:
            case 10:  mensagem = "pertence ao conjunto {6, 7, 10}";
                      break;
            case 20:
            case 21:  mensagem = "pertence ao conjunto {20, 21};
                      break;
            default:  mensagem + "não pertence a nenhum conjunto";
            }
    alert(mensagem);
</script>
```

A instrução switch - observações

- ***Comando break***: o comando break faz-se necessário, pois caso não esteja presente, o teste fará o interpretador entrar no primeiro case válido e, após este, irá percorrer todos os outros cases, até o final do laço ou até encontrar outro break.
- ***Chaves***: as chaves são obrigatórias dentro da estrutura;
- ***Default***: é opcional, e não necessita do comando break dentro dele. Todos os comandos dentro de default serão executados caso o valor sendo comparado não coincidir com nenhum valor anterior;
- Qual o tipo de dado na cláusula switch que o JavaScript permite usar? O varíavel a ser testada pode ser qualquer expressão que resulte em um tipo de dado ***numérico*** ou ***string*** ou ***booleano***.

Estruturas de repetição - while

- Esta estrutura permite a repetição de uma ou mais instruções enquanto a condição lógica sendo testada resultar verdadeira. Exemplo:

```
<script>
    var contador = 0;
    while(contador <= 10)
        {
        document.write("<h1>" + contador + "</h1>");
        contador++;
        }
</script>
```

- ***Observação***: cuidado ao usar laços de repetição. Se a expressão que você utilizar para controlar o laço (em nosso exemplo, contador <= 10) nunca chegar a retornar um valor falso, o laço será **executado para sempre**, originando um **looping eterno**. Em nosso exemplo, incrementar a variável contador com contador++ é fundamental.

Laço for

- Expressão geral:

```
for(inicialização; teste; incremento)
   {
   Comando1;
   Comando2;
   ComandoN;
   }
```

- Exemplo: fazer a mesma repetição do eslaide anterior, no formato for

```
for(var contador = 0; contador <= 10; contador++)
   document.write("<h1>" + contador + "</h1>");
```

- Note que o uso da variável contadora já está presente no cabeçalho da própria estrutura. Novamente, chamamos a atenção para a condição de teste. Se ela nunca chegar a um valor falso, o laço executará um looping eterno.

Laço for

- Ainda em relação ao laço for, ele permite que mais de uma variável seja inicializada ou incrementada dentro do cabeçalho da estrutura. É possível, também, em construções mais complexas, utilizarmos **mais** de uma condição lógica dentro do teste. Exemplo:

```
<script>

   for(var i=0, j=10; i<10 && j>i; i++, j--)
      {
      document.write("i = " + i + "<br />");
      document.write("j = " + j + "<br />");
      soma = i * j;
      document.write("soma = " + soma + "<br />");
      }
</script>
```

Vetores em JavaScript

- São coleções de dados armazenadas dentro de uma única variável ou objeto.
- ***Criação de um vetor - há diversas formas:***

```
var c = new Array(); //cria um vetor vazio e
  indefinido

var c = []; //um vetor vazio e indefinido –
  declaração literal

var c = new Array(6);//cria um vetor de 6 posições
  com valores indefinidos
```

- Os valores armazenados dentro de cada posição do vetor podem ser *inteiros, reais, texto, valores booleanos, outros vetores e objetos;*
- Um mesmo vetor pode armazenar tipos de dados diferentes.

Populando um array

- Duas maneiras: com o construtor Array() ou de forma literal. Exemplo:
- **Construtor Array():**

```
var vetor = new Array("Pedro", "Maria", 56, false);

var matriz = new Array(new Array("Pedro", 19), new
        Array("Maria", 61));
```

- **Declaração literal:**

```
var vetor[0] = "Pedro";
vetor[1] = "Maria";
vetor[2] = 56;  ou
vetor[3] = false;
```

```
var vetor = ["Pedro", "Maria", 56, false];
```

```
var matriz = [["Pedro", 19], ["Maria", 61]]; ou
matriz[0] = new Array("Pedro", 19);
matriz[1] = new Array("Maria", 61);
```

Comprimento máximo de um vetor

- Em JavaScript, um vetor pode ter um comprimento que vai desde 0 até $2^{32} - 1$, isto é, **4.294.967.295** itens. Este limite é imposto pelo tipo de dado inteiro, que, em JavaScript, tem este valor máximo;

- Cada elemento, dentro do vetor, pode ter um tamanho que é limitado apenas pela quantidade máxima de memória RAM que o navegador tem a sua disposição.

Matrizes em JavaScript

- Em JavaScript, uma matriz nada mais é do que um vetor cujos elementos são também vetores. Desta forma, poderíamos definir a matriz de duas dimensões dada pela tabela abaixo da seguinte forma:

MARCA	TAMANHO PEQUENO	TAMANHO MÉDIO	TAMANHO GRANDE
Camisa Calvin Klein	R$ 350,00	R$ 400,00	R$ 475,78
Camisa Ferrari	R$ 630,20	R$ 720,50	R$ 812,00
Camisa Lacoste	R$ 240,00	R$ 320,00	R$ 370,00

```
var camisaCal = [];
camisaCal[0] = 350;
camisaCal[1] = 400;
camisaCal[2] = 475.78;

var camisaFer = [];
camisaFer[0] = 630.20;
camisaFer[1] = 720.50;
camisaFer[2] = 812;
```

```
var camisaLac = [];
camisaLac[0] = 240;
camisaLac[1] = 320;
camisaLac[2] = 370;

var camisas = [];
camisas[0] = camisaCal;
camisas[1] = camisaFer;
camisa[2] = camisaLac;
```

Matrizes – outra forma de criação

- Podemos criar a mesma matriz da página anterior da seguinte maneira:

```
var camisas = [[350, 400,  475.78],
               [630.20,  720.50,  612],
               [240, 320, 370]]; OU

var camisas = new Array(
                    new Array(350,400,475.78),
                    new Array(630.20,720.50,612),
                    new  Array(240,320,370)
                    );
```

Percorrendo os elementos da matriz ou vetor

- Poderíamos acessar todos os elementos da matriz do exemplo anterior por meio de dois laços (for ou while). Exemplo:

```
for(var i = 0; i < camisas.length; i++)
   for(var j = 0; i < camisas[i].length; j++)
    document.write("<p>" + camisas[i][j] + "</p>");
```

Arrays com índices associativos

- Apesar de a linguagem JavaScript não suportar nativamente índices associativos com vetores, é possível utilizá-los como objetos. A linguagem lança mão da implementação de objetos para tratar os índices como sendo as propriedades do objeto Array. Veja a tabela abaixo:

EQUIPAMENTO	PREÇO
Impressora	R$ 110,14
Processador	R$ 350,00
Placa de vídeo	R$ 640,00

- Em JavaScript, teríamos:

```
var   compra = [];
compra["Impressora"]      =  110.14;
compra["Processador"]     =  350.00;
compra["Placa de vídeo"] =  640.00;
```

Percorrendo vetor com índices associativos

- Podemos fazer como se segue, tomando como exemplo o vetor criado na página anterior:

```
for(var equipamento in compra)
 {
 document.write("Equipamento: " + equipamento + "<br>");
  document.write("Prreço: " + compra[equipamento]);
  }
```

- A variável equipamento armazena o índice;
- a referência `compra[equipamento]` acessa e escreve o conteúdo do vetor naquela posição.

NOTA IMPORTANTE: vetores com índice associativo não aceitam nenhum dos métodos e propriedades mostrados nos eslaides a seguir.

Propriedades do objeto Array

Propriedade	O que faz	Exemplo

Array.length	Indica o número de elementos em uma array. Pode ser usado para diminuir o comprimento de um objeto já definido. Não pode aumentar.	nivel = ["baixo","medio","alto","muito alto"];
		
		nivel.length = 3; //reduz o tamanho de 4 para 3

Métodos do objeto Array

Método	O que faz	Exemplo

Array.join(sep)	Junta seqüencialmente os elementos de uma array usando o caractere (ou caracteres) dados por *sep*. Se ele não é indicado, usa a vírgula como default.	nivel = ["baixo","medio","alto"];
		
		str_1 = nivel.join(); //resulta "baixo,médio,alto" em str_1
		
		str_2 = nivel.join(", "); //resulta "baixo, médio, alto" em str_2
		
		str_3 = nivel.join("/"); //resulta "baixo/médio/alto" em str_3

Métodos

Método	O que faz	Exemplo

Array.pop()	Remove e retorna o último elemento de uma array, reduzindo o seu tamanho.	nivel = ["baixo","medio","alto","muito alto"];
		
		eliminado = nivel.pop; //a variável *eliminado* contém "muito alto"

Array.push(elem1, elem2, elem3,...)	Adiciona um ou mais elementos ao final de uma array, retornando o seu novo tamanho.	a variável *novo* será 4 e os dois elementos serão adicionados
		
		nivel = ["baixo","medio"];
		novo = nivel.push("alto","muito alto");

Métodos

Método	O que faz	Exemplo

Array.reverse()	Inverte a ordem dos elementos. O primeiro se torna o último e o último se torna o primeiro.	nivel[0], nivel[1] e nivel[2] serão, respectivamente, "alto", "médio" e "baixo":
		
		nivel = ["baixo","medio","alto"];
		nivel.reverse();

Array.shift()	Remove o primeiro elemento e retorna esse elemento, reduzindo o tamanho da array.	a variável *removido* terá "muito baixo" e o vetor *nivel* não terá mais esse elemento:
		
		nivel = ["muito baixo","baixo","medio","alto"];
		removido = nivel.shift();

Métodos

Método	O que faz	Exemplo

Array.slice(inicio, [fim])	O parâmetro [fim] é opcional. Copia uma parte do array e retorna um novo array com essa parte. O parâmetro inicio é o índice inicial a partir do qual começa a extração. Os elementos são extraídos até, **mas não incluindo**, o índice *fim*. Se este não é indicado, a operação se dá até o final da seqüência.
	
	A função não altera o objeto original. Os valores são copiados para um novo vetor.
		escreve: "baixo,médio,alto":
		
		nivel = ["muito baixo","baixo","medio","alto","muito alto"];
		
		document.write(nivel.slice(1,4));

Métodos

Método	O que faz	Exemplo

Array.sort(funcao)
Ou
Array.sort()
	Faz uma ordenação em ordem alfabética dos valores do array. Se os valores forem numéricos , os caracteres são tratados como texto. Neste caso, a ordenação por valor numérico não funcionará.	Para comparar números, a função pode ser simplesmente:
		
		function comparar(a, b){
		  return a - b;
		}
		num = ["11", "10", "7", "8", "9"];
		
		document.write(num.sort()); //escreve 10,11,7,8,9
		
		document.write("
		");
		
		document.write(num.sort(comparar)); //escreve 7,8,9,10,11

Método	O que faz	Exemplo

Array. splice(iniNdx, quant, [elm1][, ..., elmN])	Muda o conteúdo de uma array, adicionando novos elementos e removendo outros.
	
	iniNdx é o índice base zero a partir do qual a operação começa.
	
	quant é um inteiro indicando o número de elementos a remover. Se é zero, nenhum elemento é removido, mas deve ser indicado pelo menos um novo elemento.
	
	elm1 ... elemN são os novos elementos a adicionar. Se não indicados, a função apenas remove elementos.
	
	Se o número de elementos a adicionar é diferente do número de elementos a remover, o tamanho da array é alterado. A função retorna um array com os elementos removidos.
		num = ["10", "20", "30", "40"];
		document.write(num);
		//escreve 10,20,30,40
		
		de_fora = num.splice(2,0,"25");
		document.write(num);
		//escreve 10,20,25,30,40
		
		de_fora = num.splice(2,1,"28");
		document.write(num);
		//escreve 10,20,28,30,40
		
		document.write(de_fora);
		//escreve 25

Métodos

Método	O que faz	Exemplo
Array.toString()	Retorna uma string com os elementos dispostos seqüencialmente e separados por vírgula.	num = ["10", "20", "30", "40"]; str = num.toString(); //str contém "10,20,30,40"

Métodos

Método	O que faz	Exemplo

Array. unshift(elm1,..., elmN)	Adiciona um ou mais elementos no início do vetor e retorna seu novo tamanho. Os parâmetros (*elm1, ..., elmN*) são os novos elementos.	nivel = ["baixo","medio","alto","muito alto"];
		var a = nivel.unshift("muito baixo");
		document.write(nivel); //escreve "muito baixo,baixo,medio,alto,muito alto"
		document.write(a); //escreve 5

Métodos

Método	O que faz	Exemplo

Array.concat(vetor1, vetor2, ..., vetorN)	Adiciona um ou mais vetores a um vetor já existente.	var nomes = ["pedro", "maria"];
		var idades = [19, 64];
		var matricula = [88742, 67773];
		
		//adicionando os dois últimos ao primeiro
		var dados = nomes.concat(idades, matricula);
		
		//adicionando o primeiro e o último ao segundo
		var dados = idades.concat(nomes, matricula);
		
		//adicionando o último ao segundo
		var dados = idades.concat(matricula);

Funções de usuário em JavaScript

- Bloco de código que é definido uma única vez, mas pode ser invocado ou executado quantas vezes quiser;
- Parâmetros ou argumentos são valores especificados no momento da chamada de uma função;
- Em JavaScript, funções são de dois tipos:
  - ✓ Funções embutidas na linguagem. Exemplo: eval(), parseInt(), sort(), etc...;
  - ✓ Funções de usuários.
- São várias as vantagens do uso de funções. Dentre elas, podemos citar :
  - ✓ a organização do código;
  - ✓ a reutilização de código;
  - ✓ facilidade na busca e detecção de erros.

Definindo uma função em JavaScript

- Faz-se através da declaração **`function`**;
- A declaração function é seguida por:
  - ✓ Nome da função (mesmas regras para criação de nomes de variáveis);
  - ✓ Zero ou mais parâmetros entre parênteses, separados por vírgula;
  - ✓ O corpo da função, com os comandos entre chaves ({ }).

Exemplo:

```
//função matemática que calcula a distância entre dois pontos:
function distance(x1, y1, x2, y2)
    {
    var dx = x2 - x1; //todas as variáveis declaradas com var e os
    var dy = y2 - y1; //parâmeteros usados aqui são locais
    return Math.sqrt(dx*dx + dy*dy);
    }
```

Retorno de valores

- Uma função em JavaScript pode retornar valores ao ponto em foi invocada, no corpo do script principal;
- A declaração para retornar valores é **`return`**;
- No momento em que uma declaração return é encontrada, a função termina sua execução e o valor de retorno é enviado para o local onde a função foi chamada. A função deixa de existir e todas as variáveis locais são excluídas;
- Se não houver nenhuma declaração return, a função executa seu código até o final e devolve, como retorno, o valor **indefinido**;

Return - exemplos

Exemplo 1	Exemplo 2	Exemplo 3

function escreveMensagem()
{
document.write("Olá");
}
 	
Neste caso, não há uma declaração return. A função retornará um valor indefinido.
	function escreveMensagem()
	{
	document.write("Olá"
	);
	return;
	}
	
	Neste caso, apesar de haver uma declaração return, nada a ela foi associado. O valor de retorno também é indefinido.
		function escreveMensagem()
		{
		return var a=10;
		document.write("Olá");
		}
		
		Neste caso, a função nuca escreverá a mensagem “Olá” e retornará o valor 10 para o ponto de chamada.

Declaração dos argumentos

- Se a invocação de uma função passar mais argumentos do que a declaração espera, a função irá ignorar os argumentos extras. Veja:

```
var soma = calculaSoma(a, b, c);//o argumento c é ignorado

function calculaSoma(a, b)
    {
    //comandos
    }
```

- Se você passar menos argumentos que o esperado, os omitidos receberão o valor indefinido. Exemplo:

```
var soma = calculaSoma(a, c);

function calculaSoma(a, b, c)//c é indefinido
    {
    //comandos
    }
```

Literais de função

- A linguagem JavaScript permite que uma função seja definida sem lhe atribuir um nome. Funções assim criadas são chamadas de literais de função. A forma de se criar uma literal de função é muito parecida com a de uma declaração de função. Veja:

```
function f(x) { return x*x; } //declaração de função
var f = function(x) { return x*x; }; //literal de função
```

- Uma literal de função, também, pode receber um nome. Exemplo:

```
var f = function fatorial(x) { if (x <= 1) return 1; else return
   x*fact(x-1); };
```

- Literais de função desempenham papel importantíssimo quando desejamos separar código javascript da marcação HTML.

Variáveis globais e locais

- De modo geral, toda variável declarada dentro de uma função precedida da palavra reservada **var** é local à função. Só existe enquanto a mesma está sendo executada. No momento em que a função termina, a variável é destruída;
- Uma variável declarada com var, mas fora de uma função, **é global a todas estas funções**, isto é, se declararmos uma variável com var ou sem var fora de uma função, e utilizarmos esta mesma variável sem var DENTRO da função, trata-se da mesma variável (ela é GLOBAL): qualquer mudança feita se reflete nesta variável fora da função. De outro modo, se utilizarmos a palavra var dentro da função, esta variável, apesar de ter o mesmo nome, é LOCAL. Significa que qualquer mudança feita dentro da função não altera o valor desta variável fora da função. De fato, são duas variáveis completamente diferentes, apesar de terem o mesmo nome.

Variáveis locais e globais –função dentro de função

```
function recebeDados(a, b, c)
  {
  var media = (a + b + c); //media é local a recebeDados, mas global a
  operacao1() e local a operacao2()

  function operacao1()
    {
    media = media / 3; //media aqui é global (sem var)
    }

  function operacao2()
    {
    var media = (4 + 6)/2; //media aqui é local (com var). De fato é
  uma variável totalmente diferente daquela presente em recebeDados
    }
  }
```

Temporizadores em JavaScript

- A função **setTimeout("<codigo javascript>", <tempo>)** executará o <codigo javascript> **após** o <tempo> definido na função.
- <codigo javascript> pode ser qualquer código, desde uma única instrução ou várias instruções separadas por ponto-e-vírgula, ou a chamada a uma função.
- <tempo> é indicado em milissegundos. Exemplo:
- setTimeout("cumprimento();", 2000) executará a função cumprimento() após 2 segundos do início da execução do script onde a função setTimeout() foi invocada.
- Este tipo de temporizador é chamado de temporizador de disparo único, pois é executado uma única vez durante a existência do script.

Temporizador de repetição

- A função **setInterval("<codigo javascrip>", <tempo>)** irá executar o código dado por <codigo javascript> a cada intervalo de <tempo>, repetidamente.
- Esta função retorna para o ponto de chamada o ID do temporizador. Exemplo:
- `var idTemporizador = setInterval("mostraHora();", 1000)` executará a função mostraHora() a cada 1 segundo.
- Para encerrar a execução de setInterval, use a função clearInterval(idTemporizador). Exemplo:
- `clearInterval(idTemporizador)`, onde idTemporizador é o nome da variável criada na chamada de setInterval().

Manipuladores de eventos

- A linguagem JavaScript utiliza um modelo de programação dirigido a eventos;
- Quando o navegador gera determinada ação, um manipulador de evento pode ser associado e invocado para tratar o evento. Geralmente, é uma função de usuário, à qual o evento está associado, que será responsável por esta tarefa;
- Deve-se destacar o fato de que nem todos os navegadores tratam a manipulação de eventos da linguagem JavaScript da mesma forma. Particularmente, o Internet Explorer apresenta incompatibilidades ou implementações particularizadas para vários destes manipuladores de eventos.

Manipuladores de eventos básicos

- Abaixo, alguns manipuladores de eventos compatíveis com ***TODOS*** os navegadores atuais;

Manipulador	Disparado quando...	Elemento suportado
onabort	Interrompe-se o carregamento de uma imagem	<img>
onchange	O elemento perde o foco e o valor do elemento mudou desde que ele ganhou o foco	<input>, <select> e <textarea>
onblur	O elemento perde o foco de entrada	<button>, <input>, <label>, <select>, <textarea>, <body> <a>
onclick	Pressiona-se e libera-se o botão esquerdo do mouse ou teclas de atalho. Retornar false cancela a ação-padrão do elemento	A maioria dos elementos
ondblclick	Duplo clique esquerdo do mouse	A maioria dos elementos

Manipulador	Disparado quando...	Elemento suportado
onerror	Ocorre um erro no carregamento da imagem	<img>
onfocus	O elemento ganha o foco de entrada	<button>, <input>, <label>, <select>, <textarea>, <body> <a>
onkeydown	Uma tecla é pressionada. Retorne false para cancelar a ação-padrão do elemento	Elementos de formulário e <body>
onkeypress	Uma tecla é pressionada e está descendo. Retorne false para cancelar a ação-padrão do elemento. Não captura as teclas Alt, Control e Shift	Elementos de formulário e <body>
onkeyup	Uma tecla é pressionada e está subindo. Retorne false para cancelar a ação-padrão do elemento	Elementos de formulário e <body>
onload	A página, uma imagem ou um frame terminou de ser carregado	<body>, <frameset>, <img>
onmousedown	O botão do mouse foi pressionado	A maioria dos elementos

Manipulador	Disparado quando...	Elemento suportado
onmousemove	O mouse é movido sobre o elemento	A maioria dos elementos
onmouseout	O mouse move-se para fora do elemento	A maioria dos elementos
onmouseover	O mouse está fora do elementos e é movido sobre o ele	A maioria dos elementos
onmouseup	O botão do mouse é liberado	A maioria dos elementos
onreset	Um formulário é recarregado (através de um botão reset ou F5). Retorne false para impedir a ação-padrão no formulário	<form>
onresize	O tamanho da janela do navegador muda	<body> e <frameset>
onselect	Um texto é selecionado	<input> e <textarea>
onsubmit	Um formulário é enviado (através da tecla ENTER ou um botão de envio). Retorne false para impedir a ação-padrão	<form>

Eventos de mouse não suportados

- Todos os elementos HTML suportam eventos de mouse, com exceção de:
- <base>, <bdo>,
, <head>, <html>, <iframe>, <meta>, <param>, <script>, <style> e <title>

Eventos compostos (em cascata) - I

- Observe que uma determinada ação do usuário sobre um elemento da página web pode desencadear mais de um evento, em sequência. Exemplo: quando o usuário clica com o mouse sobre um botão de envio em um formulário, temos os seguintes eventos disparados, nesta ordem:

  1. **onmousedown;**
  2. **onmouseup;**
  3. **onclick;**
  4. **onsubmit.**

- Outra coisa: se o botão ***submit*** tiver o foco e o usuário pressionar a tecla ***enter***, os eventos disparados são ***onclick*** e ***onsubmit***, nesta ordem. Isso equivale a clicar com o mouse sobre o botão, mas sem acionar os eventos onmouseup e onmousedown.

Eventos compostos (em cascata) - II

- Quando você desloca o foco sobre um link, usando a tecla ***tab*** e, em seguida, pressiona a tecla ***enter***, essa ação dispara o evento ***onclick*** sobre o link;
- Quando o usuário pressiona o botão esquerdo do mouse sobre um elemento da página, sem soltá-lo, e arrasta este elemento para outro local, o evento sendo disparado é o ***onmousedown***;
- Da mesma forma, quando o usuário termina de arrastar um item na página e libera o botão esquerdo do mouse, o navegador dispara o evento ***onmouseup***;
- Quando o usuário pressiona uma tecla e a libera em seguida, ocorre a seguinte cadeia de eventos:
  1. **onkeydown;**
  2. **onkeyup;**
  3. **onkeypress.**

Manipuladores de eventos como atributos

- Quando os manipuladores de evento são utilizados dentro da própria tag HTML, eles funcionam como atributos da tag. Veja:

```
<button type="submit" onclick="validaDados();"> ou

<button type="button" value="Clique-me!"
  onclick="alert('Botão clicado!');" />
```

Manipuladores de eventos como propriedades do objeto

- O JavaScript permite referir-se a um manipulador de evento como se o mesmo fosse uma propriedade do objeto HTML ao qual ele está associado. Exemplo:

```
<form name="f1">
   <input name="b1" type="button" value="clique-me">
 </form>
```

Dentro do script, poderíamos referenciar o botão do formulário utilizando o name do formulário e o name do botão. O formulário funciona como uma propriedade do objeto document e o botão de radio seria uma propriedade do objeto formulário, como no exemplo abaixo:

```
document.f1.b1.onclick=function( ) { alert("Obrigado!"); }; OU
function agradece( ) {document.f1.b1.value = "Obrigado!";}
document.f1.b1.onmouseover = agradece;
```

Manipuladores de eventos como função. Cuidado!

- Ao atribuir a chamada de uma função a um manipulador de evento que está associado a um objeto como sua propriedade, **NUNCA** use parênteses após o nome da função. Isso é porque você está querendo atribuir ao evento não o resultado da execução da função, mas sim o próprio código da função, para que o manipulador o execute no momento em que ele for acionado. Lembra da utilidade de literais de função? Veja:

Errado:

```
function cumprimenta()
    {
    paragrafo.innerHTML = "Olá! Seja bem-vindo!";
    }
paragrafo.onclick = cumprimenta();//erro
```

Correto:

```
function cumprimenta()
    {
    paragrafo.innerHTML = "Olá! Seja bem-vindo!";
    }
paragrafo.onclick = cumprimenta; //sem parênteses
```

Manipuladores de evento como funções

- **Vantagens:**

1. Reduz a mistura do código JavaScript com o HTML.
2. Favorece um código mais modular, mais legível, mais limpo e de fácil manutenção e modificação;
3. Permite criar eventos dinamicamente, e não apenas eventos estáticos amarrados às tags HTML e já pré-definidos dentro destas tags.

- **Desvantagem:**

1. Uma pequena desvantagem desta abordagem é que, como ela separa o manipulador do seu elemento HTML associado, corre-se o risco de se executar o código JavaScript **ANTES** que a página seja carregada e, neste caso, o elemento ainda não estará definido, o que causará um erro no script. Solução? Chamar o código JavaScript ao final da seção <body>.

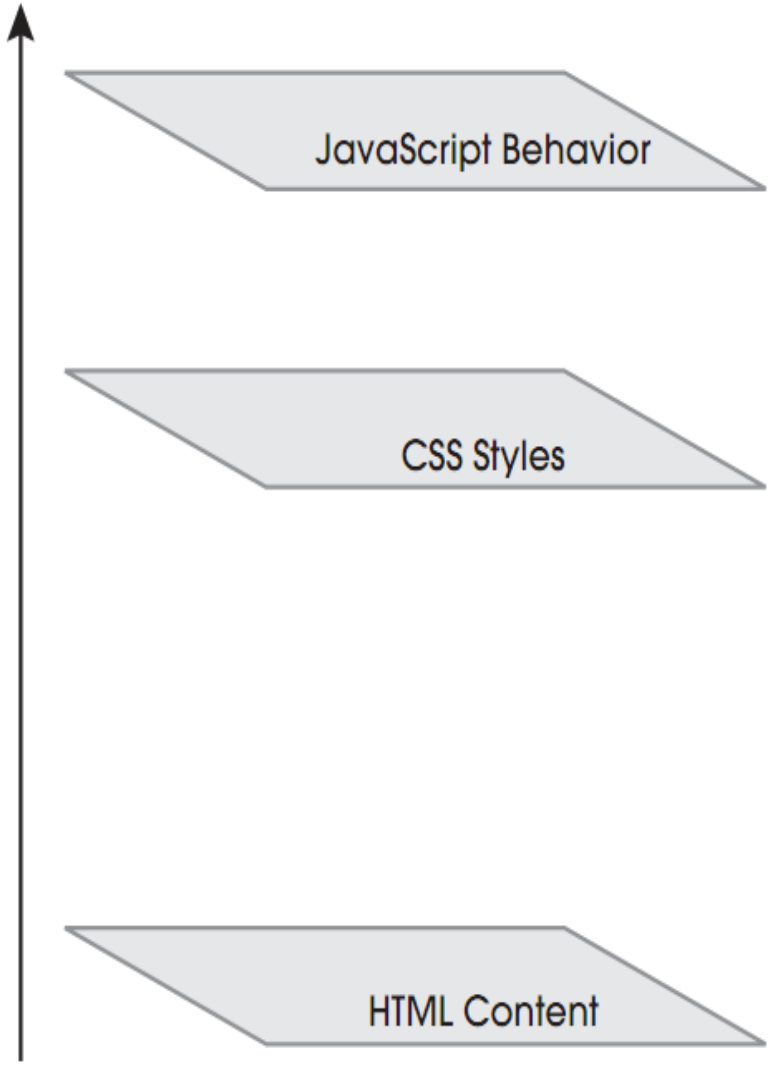
Camadas de um documento HTML

- A DHTML permite que um documento para a web seja decomposto em três conjuntos fundamentais de dados, que englobam cada um dos elementos relacionados, produzindo o que chamamos de camadas da aplicação web:

  1. ***Comportamento:*** **JavaScript;**
  2. ***Apresentação:*** **CSS;**
  3. ***Estruturação:*** **HTML**

- Um dos objetivos de um documento para a web bem projetado é agrupar cada um destes segmentos de dados em arquivos separados. Essa abordagem é chamada de **JavaScript não intrusivo**. Sendo assim, aconselha-se, sempre que possível, a utilização de arquivos externos, tanto arquivos .css quanto arquivos .js

Esquema gráfico para separação de camadas

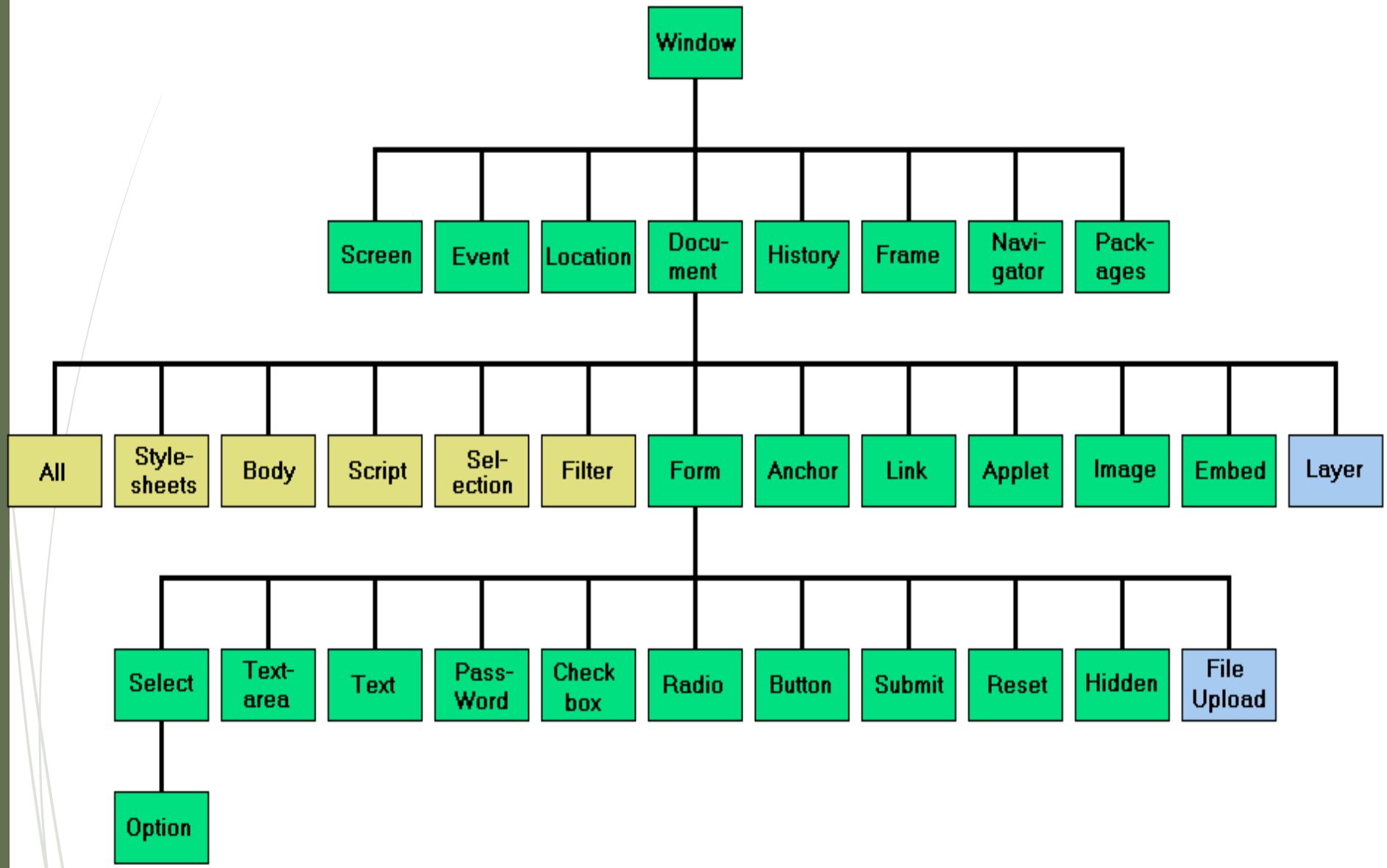



```
<input type = "text"
       id = "email"
       onChange="checkEmail()">
```

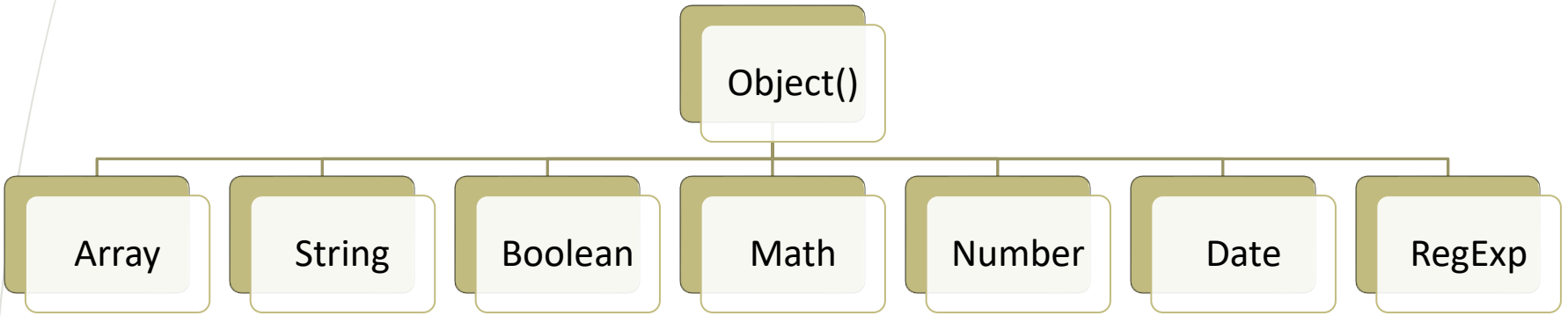
```
body { background-color:silver; }
p.first { font-family:"sans serif"; }
h1, h2, h3 { color: darkblue; }
```

```
<html>
  <head>
    <title>HTML Page</title>
  </head>
  <body>
    <h3>Hello, world!</h3>
  </body>
</html>
```

Objetos JavaScript relacionados a uma página web (DOM-W3C)



Objetos JavaScript do núcleo da linguagem (core - ECMA)



Propriedades e métodos para manipulação do DOM

Propriedades dos nós	
childNodes	Vetor de nós-filhos
firstChild	O primeiro nó-filho
lastChild	O último nó-filho
nodeName	Nome do nó
nodeType	Tipo do nó
nodeValue	Valor do nó. Geralmente, o texto entre as tags
nextSibiling	Próximo nó-irmão
previousSibiling	Nó-irmão anterior
parentNode	Nó-pai

Propriedades e métodos para manipulação do DOM

Métodos dos nós	
AppendChild	Adiciona um nó-filho ao pai
HasChildNodes	Verdadeiro se o nó tem filhos
RemoveChild	Apaga o nó-filho

Propriedades e métodos para manipulação do DOM

Alguns métodos do objeto document	
CreateAttribute()	Cria um novo atributo para o elemento
CreateElement()	Cria um novo elemento. Não insere na árvore do DOM na memória RAM usada pelo navegador
CreateTextNode()	Cria um conteúdo de texto entre as tags
getElementsByTagName()	Cria um vetor com todos os elementos da tag especificada
getElementById()	Acessa o elemento através de seu id
getElementsByClassName()	Cria um vetor de elementos que possuam determinada classe
getElementsByName()	Cria um vetor com todos os elementos de um determinado nome de tag
querySelector()	Toma um seletor CSS e retorna o primeiro elemento que se encaixa na seleção
querySelectorAll()	Toma um seletor CSS e retorna um vetor de todos os elementos que se encaixam na seleção

Propriedades e métodos para manipulação do DOM

Algumas propriedades JavaScript para objetos que representam qualquer elemento HTML	
Objeto.ClassName	Retorna ou atribui o nome da classe (ou classes) que pertencem ao elemento
Objeto.id	Retorna ou atribui o valor do atributo id do elemento
Objeto.tagName	Retorna o nome da tag (elemento) à qual o objeto JavaScript faz referência