

Programação Web

- ✓ JavaScript
- ✓ Objeto Math
- ✓ Função toFixed
- ✓ Objeto String
- ✓ Objeto Date
- ✓ Objeto Event
- ✓ Vetores
- ✓ Métodos push, pop, shift e unshift

- ✓ Estrutura de repetição for...of
- ✓ Método map
- ✓ Método forEach
- ✓ Propriedade innerHTML e innerText
- ✓ Alterando o CSS em JS
- ✓ Funções setTimeout e setInterval
- ✓ Exemplos
- ✓ Exercícios

Math é um objeto que armazena propriedades e métodos especializados para a realização **de cálculos matemáticos**. Não é possível criar uma variável com esse tipo utilizando a palavra new, basicamente utilizamos o objeto com suas respectivas propriedades e métodos.

Sintaxe geral:

Math.propriedade

ou

Math.método();

Math.**PI**: constante grega π . Valor = 3.141592....

Math.**SQRT2**: raiz quadrada de 2. Valor = 1.4142....

Math.**pow(x, y)** : x elevado a y. (atualmente temos o operador ****** que realiza o mesmo cálculo)

Math.**sqrt(x)** : raiz quadrada de x.

Math.**abs(x)** : valor absoluto de x.

Math.round(x) : arredonda o valor de x para cima ou para baixo dependendo do valor decimal após a vírgula. Abaixo de 0.5 é para baixo e para valor iguais ou superiores a 0.5 arredonda para cima.

Math.min (x, y, ..., n): mínimo entre os valores informados.

Math.max (x, y, ..., n): máximo entre os números informados.

Math.random(): gera um número aleatório entre 0 e 1.

Math.floor(x): arredonda um número sempre para baixo. Ex: `Math.floor(3.4) = 3`

Math.ceil(x): arredonda um número sempre para cima. Ex: `Math.ceil(3.2) = 4`

Em muitos casos realizamos contas e temos como retorno números com diversas casas decimais, nestes caso, se quisermos apresentar os valores com menos casas, podemos usar a função **toFixed(qtd_digitos)**

A função faz o retorno em formato String.

```
let num = 3.14854517
console.log(num.toFixed()) // Retorna '3', sem a parte decimal
console.log( num.toFixed(1)) // Retorna '3.1'
console.log( num.toFixed(2)) // Retorna '3.15'
```

Strings são criadas automaticamente quando uma variável recebe um valor literal. Esta variável pode então acessar propriedades e métodos (funções) diretamente associadas ao objeto String.

A criação de uma string pode ser feita quando atribuímos um valor literal para uma variável ou através do objeto String.

`let x = "algum texto" ou let y = new String("algum texto")`

Para acessar as propriedade ou métodos de uma string, a sintaxe geral é:

variável.propriedade

ou

variável.método();

Caracteres Especiais que podem ser utilizados em strings:

<code>\"</code> - aspas	<code>\n</code> - pula linha
<code>\'</code> - apóstrofe	<code>\r</code> - carriage return
<code>\b</code> - backspace	<code>\f</code> - form feed
<code>\t</code> - TAB	

`variável_string.length`: retorna o número de caracteres da string.

`variável_string.charAt(índice)`: retorna o caractere localizado no índice, este método é antigo, podemos usar simplesmente `[]` para acessar uma posição de uma string. Ex: `variável_string[3]`.

`variável.charCodeAt(índice)`: retorna o código ASCII correspondente ao caractere localizado no índice.

`variável_string.indexOf(string [,índice])` : busca a primeira ocorrência de uma string a partir de um índice.

`variável_string.lastIndexOf(string [,índice])`: busca a última ocorrência de uma string.

`variável.substring(índice1, índice2)`: extrai uma string partindo do índice1 até o índice2.

`variável_string.toLowerCase()`, `variável_string.toUpperCase()`: Converte para minúscula e para maiúscula.

`variável_string.split(separador)`: separa uma string pelo separador indicado no parâmetro e cria um vetor com esses dados separados.

Date é um objeto que permite manipular datas e horários.

Sintaxe geral:

```
variável = new Date(); // utiliza a data-hora atual do sistema
```

ou

```
variável = new Date("mês dia ano horas:mins:secs")
```

```
//cria o objeto predefinido
```

ou

```
variável = new Date(ano, mês, dia, horas, minutos, secs)
```

```
//cria a data-hora conforme os parâmetros
```

ou

```
variável = new Date( milisegundos)
```

```
//o tempo transcorrido a partir da data base (01/jan/1970)
```


Propriedades:

seconds e **minutes**: 0 a 59.

hours: 0 a 23.

day: 0 a 6 (dia da semana (0=Domingo, 1=Segunda, 2=Terça, ...)).

date: 1 a 31 (dia do mês).

month: 0 (janeiro) a 11 (dezembro).

fullyear: quatro dígitos

milliseconds: valores de 0 a 9999

Métodos:

variável.getPropriedade (); // Obtém o valor da propriedade

ou

variável.setPropriedade (argumento); // Muda o valor da propriedade

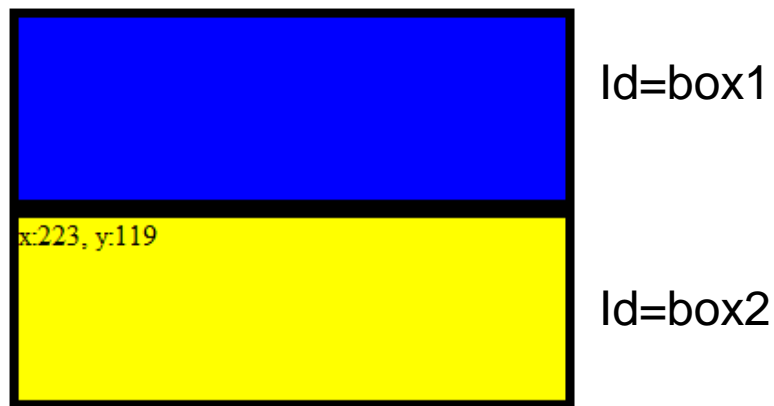
Sempre que ocorre um evento, um objeto event é criado.

Propriedade type, retorna o tipo do evento que ocorreu. Todos os navegadores atuais suportam.

No IE podemos ter uma variação: tem o event ou o window.event

Exemplo:

```
let btn = document.querySelector("#botao");  
btn.addEventListener("click", function(event) {  
    alert(event.type);  
});
```



Eventos (box 1)

- click - cor de fundo verde
- dblclick - cor de fundo azul

Eventos box 2

- mouseover - cor de fundo vermelho
- mousedown - cor de fundo preto
- mouseup - cor de fundo branco
- mouseout - cor de fundo amarelo
- mousemove - mostra x,y do mouse dentro da DIV

```
let elemento1;  
let elemento2;
```

```
//associar os eventos aos elementos
```

```
function trataEventos1(event)  
{  
    tratar os eventos do box1  
}  
function trataEventos2(event)  
{  
    tratar os eventos do box2  
}
```

O objeto event possui as principais propriedades abaixo:

keyCode

retorna o código (número) de qualquer tecla (funciona para os eventos keydown e keyup)

charCode

retorna o código (número) de uma tecla alfanumérica (funciona para o evento keypress)

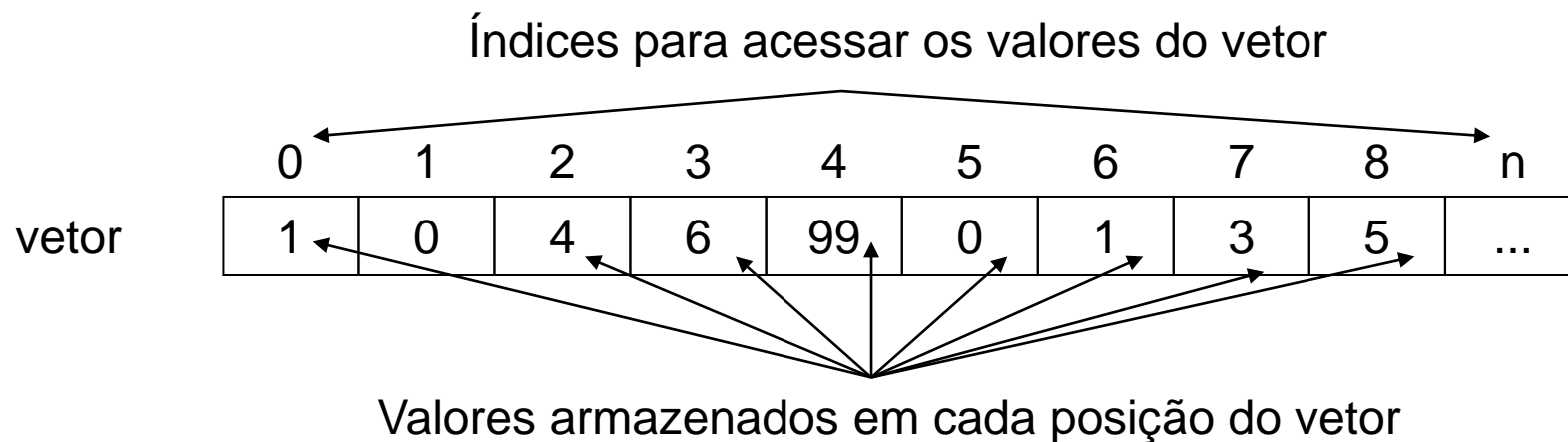
```
let elemento = document.body;  
elemento.onkeydown = trataEvento1;  
elemento.onkeyup = trataEvento1;  
elemento.onkeypress = trataEvento2;
```

```
function trataEvento1(event){  
    document.querySelector("#box1").innerHTML = "keyCode<br>" + event.keyCode;  
}  
  
function trataEvento2(event){  
    document.querySelector("#box2").innerHTML = "charCode<br>" + event.charCode;  
}
```

keyCode	charCode
65	97

Vetores são vistos como coleção de valores, onde através de um único nome de variável, temos várias posições para guardar informações.

Em uma representação gráfica podemos visualizar um vetor da seguinte forma:



OBS: o primeiro índice de um vetor começa sempre em 0 (zero)

Para criar um vetor em JavaScript utilizamos o objeto `Array`. Este objeto além de criar um vetor em memória também nos fornece diversos métodos para manipular o vetor criado. Esses métodos serão listados posteriormente.

Podemos criar um vetor de duas formas:

```
let vetor = new Array();
```

```
let vetor = [ ];
```

```
<script>
```

```
//vetor sem tamanho definido, inicialmente vazio:
```

```
let vetorA = new Array();
```

```
let vetorB = [ ];
```

```
//vetor com posições e valores definidos:
```

```
let vetorC = new Array("maça", "banana", "morango");
```

```
let vetorD = [ "maça", "banana", "morango" ];
```

```
let vetorE = [ 12, 6, 21, 5, 7, 209, 3 ];
```

```
</script>
```

OBS: em todos os casos acima, podemos inserir mais posições sempre que necessário. **O Vetor não tem um tamanho fixo.**

Para visualizarmos ou atribuírmos um valor para um vetor devemos acessar cada posição através do índice. Este índice deve ficar entre colchetes [...]

Exemplo:

```
let frutas = [];  
frutas[0]= "maça";  
frutas[1]= "banana";  
frutas[2]= "morango";  
alert(frutas[2]);
```

```
let data_hora = new Date();  
let meses = ["JAN", "FEV", "MAR", "ABR", "MAI", "JUN", "JUL", "AGO", "SET", "OUT", "NOV", "DEZ"];  
alert(meses[data_hora.getMonth()]);
```

Em quase todos os casos em que trabalhamos com vetores, utilizamos uma estrutura de repetição para acessar suas posições. A mais comum seria a estrutura for, porém também podemos usar o comando while.

Exemplo

exemplo10.html

Sem o uso de repetições (for, while...)

```
let frutas = [ ];  
frutas[0]="maça";  
frutas[1]="banana";  
frutas[2]="morango";  
console.log(frutas[0]);  
console.log(frutas[1]);  
console.log(frutas[2]);
```

exemplo11.html

Com uso de repetições (comando for)

```
let i;  
let frutas = [ ];  
frutas[0]="maça";  
frutas[1]="banana";  
frutas[2]="morango";  
for (i=0; i<=2; i++){  
    console.log(frutas[i]);  
}
```

length

Retorna o número de elementos de um vetor

- Ex:

```
let vetor1 = [ ];  
vetor1[0] = "maça";  
vetor1[1] = "banana";  
alert("Tamanho: " + vetor1.length);
```

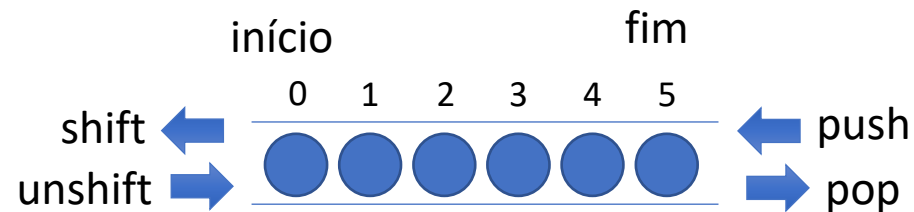
Podemos usar o vetor em JS como uma estrutura de dados do tipo fila ou do tipo pilha, dependendo do caso, existem métodos diferentes para algumas ações. A diferença é somente no uso do método.



Outro método disponível é o **unshift**, neste caso, ele permite inserir no início da vetor .

```
let frutas = ["Maça", "Laranja"];  
console.log(frutas);  
  
frutas.push("Morango");  
frutas.push("Uva", "Abacaxi"); //podemos inserir mais de 1 elemento  
console.log(frutas);  
  
console.log(frutas.pop()); //remove o último elemento Abacaxi  
console.log(frutas);  
  
console.log(frutas.shift()); //remove o primeiro elemento Maça  
console.log(frutas);  
  
frutas.unshift('Melão'); //adiciona no início  
frutas.unshift("Limão", "Damasco");//podemos adicionar mais de 1 elemento no início  
console.log(frutas);
```

Métodos mais lentos
devido ao fato de
precisarem
reorganizar os índices
do vetor.



Métodos mais rápidos já
que não precisam
reorganizar os índices do
vetor.

Como demonstramos anteriormente, para percorrer todas as posições do vetor, devemos usar repetições.

Podemos usar a estrutura **for** ou então o laço chamado **for...of**.

O **for** funciona em todos os navegadores e seu contador nos permite acessar o vetor pelo índice.

O **for...of** é uma sintaxe mais moderna, contudo não acessa o vetor pelo índice, ele basicamente percorre os elementos diretamente um após o outro.

```
let campo = document.querySelector("#res");

let meses = ["JAN", "FEV", "MAR", "ABR", "MAI", "JUN",
             "JUL", "AGO", "SET", "OUT", "NOV", "DEZ"];

campo.innerHTML = "For normal<br>";
for (let i = 0; i < meses.length; i++) {
    campo.innerHTML += meses[i] + "<br>";
}

campo.innerHTML += "<hr>";
campo.innerHTML += "For..of<br>";
for (let mes of meses) {
    campo.innerHTML += mes + "<br>";
}
```

concat() - Retorna a junção (cópia) de dois ou mais vetores

Ex:

```
//retorna a junção do vetor1 com o vetor2
```

```
vetor1.concat(vetor2);
```

```
//retorna a junção do vetor1 com o vetor2 e vetor3
```

```
vetor1.concat(vetor2, vetor3);
```

reverse() - inverte a ordem dos elementos no vetor

Ex:

```
vetor1.reverse();
```

sort() - ordena os elementos do vetor

Ex:

```
vetor1.sort()
```

toString() - converte e retorna um vetor em String

Ex:

```
vetor1.toString();
```

splice(início, fim) – seu uso mais comum é remover um ou mais elementos

Ex:

```
vetor1.toString();
```


vetor.indexOf(item, início) - procura o item a partir do índice informado, retorna o índice onde o elemento foi encontrado, caso contrário retorna -1.

vetor.lastIndexOf(item, início) – semelhante ao anterior, mas procura a última ocorrência do elemento no vetor retornando sua posição ou -1 se não encontra.

vetor.map(função) – aplica uma função a cada posição do vetor e retorna um vetor como resultado

```
let result = vetot.map(function(item, index, array) {  
    //retorna um novo item para cada valor  
});
```

O método `forEach` permite executarmos uma função para cada elemento do vetor.

```
vetor.forEach(function(item, index, array) {  
    // ... Função para manipular o item  
});
```

```
let vetor1 = ["México", "Brasil", "Itália", "Espanha", "Argentina", "Chile"];
```

```
vetor1.forEach(alert);
```

```
let campo = document.querySelector("#res");  
vetor1.forEach( (item, index) => {  
    campo.innerHTML += `País: ${item} Índice: ${index}<br>`;  
});
```

innerHTML: retorna ou modifica o conteúdo HTML de um elemento

innerText: retorna ou modifica o conteúdo TEXTO de um elemento

Sintaxe:

para capturar o valor (HTML) do elemento:

`variável = document.getElementById(ID).innerHTML` ou `variável = document.querySelector(ID).innerHTML`

para capturar o valor (TEXTO) do elemento:

`variável = document.getElementById(ID).innerText` ou `variável = document.querySelector(ID).innerText`

para alterar o valor do elemento inserindo comandos HTML com o conteúdo:

`document.getElementById(ID).innerHTML="valor"` ou `document.querySelector(ID).innerHTML`

para alterar o valor do elemento inserindo apenas Texto:

`document.getElementById(ID).innerText="valor"` ou `document.querySelector(ID).innerText`

Exemplo 18

Título:

Cria tabela:

Linhas: Colunas:

Linha: 1 - Coluna:1	Linha: 1 - Coluna:2	Linha: 1 - Coluna:3	Linha: 1 - Coluna:4
Linha: 2 - Coluna:1	Linha: 2 - Coluna:2	Linha: 2 - Coluna:3	Linha: 2 - Coluna:4
Linha: 3 - Coluna:1	Linha: 3 - Coluna:2	Linha: 3 - Coluna:3	Linha: 3 - Coluna:4

No material de referência de CSS disponibilizado em aulas passadas, encontrará a sintaxe em JavaScript para cada atributo do CSS.

Sintaxe padrão

```
elemento.style.atributo = "valor"
```

Exemplo

```
document.querySelector("#caixa").style.color="red";
```

```
...  
<body>  
  <div id="caixa"> Texto da DIV </div>  
</body>  
...
```

Utilizando a sintaxe padrão `elemento.style.atributo = "valor"` podemos alterar ou recuperar diferentes atributos (estilos) de um elemento.

Exemplo

```
document.querySelector("#divA").style.color= "blue";  
document.querySelector("#divA").style.display = "block";  
document.querySelector("#divA").style.width = "900";
```

```
document.querySelector("#divB").style.color= "yellow";  
document.querySelector("#divB").style.display = "none";
```

```
<body>  
  <div id="divA"> Texto desta div </div>  
  <div id="divB"> Texto desta outra div </div>  
</body>
```

Tamanho da Fonte=

Cor da fonte:

Cor de fundo:

Tamanho da div (width, exs: 640px 50% auto 200px)=

Lorem, ipsum dolor sit amet consectetur
adipiscing elit. Ipsam perferendis nostrum
dicta ad temporibus iure perspiciatis quasi
ipsum optio? Quos, accusantium! Impedit
sequi tempora illum aliquam corrupti deleniti
saepe nihil! Dicta reiciendis voluptates
delectus quod quidem, commodi atque
quisquam repudiandae? Reprehenderit porro
culpa aspernatur magnam fugit, optio est
cupiditate rem iusto blanditiis nobis
voluptatibus iure. Eos voluptates voluptatibus
itaque esse?

exemplo19.html

São funções para manipularmos o tempo, ou seja, definimos tempo para a execução de ações determinadas.

Sintaxe:

```
setTimeout(funcao, intervalo_em_milisegundos);  
setInterval(funcao, intervalo_em_milisegundos);
```

As duas funções anteriores irão chamar uma segunda funcao() passada por parâmetro, no intervalo especificado em milisegundos. A função que será chamada neste caso não deve ter os parênteses.

A setTimeout() chama a função uma única vez. Enquanto a setInterval() chama a função "infinidamente" sempre no mesmo intervalo de tempo.

Para finalizar as funções usa-se clearTimeout() ou clearInterval() respectivamente. Passando como parâmetro o nome do seu intervalo, retornado quando chamada alguma das funções setInterval() e setTimeout().



exemplo20.html



A screenshot of a web browser window displaying the number '16' in a large, red, serif font. The browser's address bar and other interface elements are not visible.

Sorteando valores, uso do objeto Math e funções setInterval e setTimeout

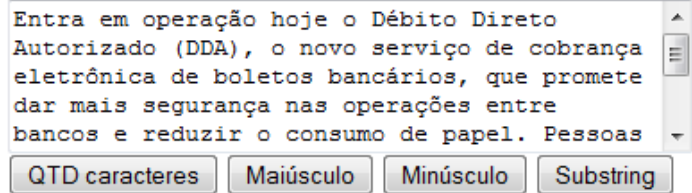
exemplo21.html



A screenshot of a web browser window showing a digital clock. The time '9:41:16' is displayed in a yellow, italicized, serif font on a black rectangular background. The browser's address bar and other interface elements are not visible.

Relógio digital, uso do objeto Date e função setInterval

exemplo22.html



A screenshot of a web browser window. It features a text area containing a paragraph of text in Portuguese: 'Entra em operação hoje o Débito Direto Autorizado (DDA), o novo serviço de cobrança eletrônica de boletos bancários, que promete dar mais segurança nas operações entre bancos e reduzir o consumo de papel. Pessoas'. Below the text area are three buttons: 'QTD caracteres', 'Maiúsculo', and 'Minúsculo'. There is also a 'Substring' button to the right of the 'Minúsculo' button. The browser's address bar and other interface elements are not visible.

Manipulando strings

exemplo23.html

<http://www.w3schools.com/js/default.asp>

GOODMAN, D. Javascript e Dhtml Guia Pratico. Rio de Janeiro: Alta Books, 2008.

NEGRINO, T.; SMITH, D. Javascript Para a World Wide Web. 4. ed. Rio de Janeiro: Campus, 2001.

SILVA, M. S. Construindo Sites Com Css e (X) Html: Sites Controlados por Folhas de Estilo em. Sao Paulo: Novatec, 2010.

MICHAEL MORRISON. **Use a Cabeça Javascript**. São Paulo: Alta Books, 2008.

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/contains
http://exploringjs.com/es6/ch_arrays.html