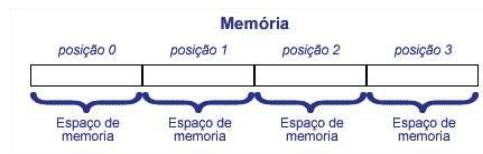


## 7.0 Array

Nesse capítulo iremos abordar de forma clara as principais características de um array; Mostrar a sua criação e manipulações possíveis; Definir arrays multidimensionais ou matrizes; Determinar formas de interações e acessos.

Um array no JavaScript é atualmente um conjunto de valores ordenado por índices. Podemos relacionar cada valor com um índice/chave, para indicar em qual posição um determinado valor está armazenado dentro do array.



Ele é otimizado de várias maneiras, então podemos usá-lo como um array real, lista (vetor), hashtable (que é uma implementação de mapa), dicionário, coleção, pilha, fila e provavelmente muito mais. Além disso, JavaScript nos oferece uma gama enorme de funções para manipulá-los.

A explicação dessas estruturas está além do escopo dessa apostila, mas todo conteúdo aqui abordado traz uma boa base para quem estar iniciando o conteúdo de array.

### 7.1 Criando um Array

Existe várias formas de construção de um array em JavaScript, veja:

- Construção simples sem dimensionamento: Quando criamos um array, mas não especificamos quantas posições ele irá possuir;
- **Construção simples com dimensionamento:** Quando criamos um array e especificamos quantas posições ele irá possuir. Estas posições são vazias.
- Construção inserindo valores: Quando criamos um array e especificamos exatamente quais valores ele irá possuir. Cada valor deve ser separado por vírgula e qualquer tipo de dado pode ser utilizado para popular nosso vetor.

Veja um exemplo de cada forma de criar um vetor/array:

```
<script type="text/javascript">
  //Sem dimensionamento
  meuVetor1 = new Array();

  //Com dimensionamento
  meuVetor2 = new Array(4);

  //Com valores inseridos
  meuVetor3 = new Array("Raquel","Fabiana","Diana","Alana");
</script>
```

Detalhe importante é que todo array não associativo começa pela chave ou índice de número 0.

A figura abaixo representa um array que tem como valor representação de cores, e possui dez posições,



cada posição representa uma cor, seu índice (chave) vai de 0 até 9. Veja:

Em código temos:

```
<script type="text/javascript">
  var cores = new Array("verde","azul","violeta","cinza",
    "vermelho","amarelo","magenta","branco","laranja","marrom");
</script>
```

### 7.2 Arrays associativos.

Os arrays associativos associa-se um determinado valor ou nome a um dos valores do array.

O array associativo usa strings como índice, onde cada string pode representar uma chave.

Observe a sintaxe:

```
<script type="text/javascript">
  var vetor = {"chave1":"valor1","chave2":"valor2","chaveN":"valorN",}
</script>
```

Observe que quando usamos arrays associativos, a compreensão é mais fácil, dando mais legibilidade ao código. Porém, não é adequado utilizá-lo dentro de um loop tradicional(while,do..while,for), por isso temos um loop especial para percorrer arrays associativos chamado de for..in.

```
<script type="text/javascript">
//Forma 01
var vetor1 = {"nome": "Valdenia", "rua": "São João", "bairro": "Messejana"};
//Forma 02
var vetor2 = new Array[3];
vetor2["nome"] = "Valdenia";
vetor2["rua"] = "São João";
vetor2["bairro"] = "Messejana";
</script>
```

Veja um exemplo de array associativo escrito de duas formas diferentes:

Umas das vantagens do array associativo é quando fazemos o acesso ao array, onde temos de forma clara e compreensível o valor que aquela chave pode conter. Como por exemplo nome, onde só vai existir o nome de pessoas. Veja abaixo um exemplo de acesso aos valores armazenados em um array dessa natureza.

#### Exemplo:

```
<script type="text/javascript">
var vetor = new Array(3);
vetor['nome'] = "Valdenia";
vetor['rua'] = "São João";
vetor['bairro'] = "Messejana";

document.writeln(vetor['nome']);
document.writeln(vetor['rua']);
document.writeln(vetor['bairro']);
</script>
```

Dessa forma podemos acessar o array. Basta determinar o nome do array e qual a chave, onde cada chave tem um valor já determinado. Resultará em um erro o uso de uma chave errada.

### 7.3 Interações

Quando falamos de interações em um array estamos dizendo o mesmo que percorrer esse array usando mecanismos da própria linguagem. Como isso as interações podem ser feitas de várias formas, mas no JavaScript podem ser iterados pelo operador FOR( chave IN array ) que já vimos anteriormente.

#### Exemplo:

```
<script type="text/javascript">
var vetor = new Array(3);
vetor['nome'] = "Valdenia";
vetor['rua'] = "São João";
vetor['bairro'] = "Messejana";

for (chave in vetor){
    document.write(chave + " = " + vetor[chave] + "<br>");
}
</script>
```

#### Resultado:

```
nome = Valdenia
rua = São João
bairro = Messejana
```

Esse tipo de interação é muito utilizado, principalmente quando temos arrays associativos.

### 7.4 Acessando um Array

Quando criamos um array temos que ter em mente que estamos criando uma variável que possui vários valores e que os mesmos podem ser acessados a qualquer momento. Cada valor está guardado em uma posição que pode ser acessada através de uma chave.

A sintaxe para acesso simplificado de um array é a seguinte:

**nome\_do\_array[chave\_de\_acesso];**

Temos que ter cuidado ao passar uma chave para o array, pois ela deve conter o mesmo nome de qualquer uma das chaves existentes no array. Caso a chave não exista, o valor não poderá ser resgatado. A sintaxe acima retorna um valor contido no array, por esse motivo temos que atribuir esse valor como mostra o exemplo abaixo:

```

<script type="text/javascript">
//Array com Valores
    var meu_array = new Array("nome","telefone","rua");
// Acessando uma posição
    var elemento_array = meu_array[1];

document.write(elemento_array);
</script>

```

**Resultado:** telefone.

## 7.5 Alterando um Array

Podemos alterar qualquer valor de um array. É muito semelhante ao acesso, onde, a diferença está na chamada do array. É nesse momento que atribuímos um novo valor.

```
nome_do_array[chave_de_acesso] = <novo_valor>;
```

Observe o exemplo abaixo:

```

251<script type="text/javascript">
252 //criando um array com valores
253 var meu_array = new Array("nome","telefone","rua","cidade");
254
255 //alterando o valor de uma posição
256 meu_array[1] = "sobrenome";
257
258 //imprime na tela.
259 for(chave in meu_array){
260     document.write(meu_array[chave]+<br>);
261 }
262 </script>

```

**Resultados:**

```

nome
sobrenome
rua
cidade

```

Vimos no exemplo anterior o valor da posição 1 do array ('telefone') foi alterada para sobrenome. Vale ressaltar que esse array tem suas chaves definidas de forma automática. A primeira posição é 0, a segunda é 1, e assim sucessivamente. Veja mais um exemplo onde alteramos o valor, mas usando o operador “+=”;

```

268<script type="text/javascript">
269 //criando um array vazio
270 var produto = new Array();
271 //adicionando valores
272 produto["nome_produto"] = "Arroz";
273 produto["valor_produto"] = 1.35;
274
275 //alterando valor do produto
276 produto["valor_produto"] += .63;
277
278 //alterando nome do produto
279 produto["nome_produto"] = " Tio João ";
280
281 //imprime na tela.
282 for(chave in produto){
283     document.write(produto[chave]+<br>);
284 }
285 </script>

```

Podemos observar que assim como as variáveis “comuns”, a forma de alterar o valor de um array é igual. A diferença está na chamada do elemento de um array, pois temos que passar a chave além do valor que queremos atribuir.

## 7.6 Arrays multidimensionais

Os arrays multidimensionais são estruturas de dados que armazenam os valores em mais de uma dimensão. Os arrays que vimos até agora armazenam valores em uma dimensão, por isso para acessar às posições utilizamos somente um índice ou chave. Os arrays de 2 dimensões salvam seus valores de alguma forma como em filas e colunas e por isso, necessitaremos de dois índices para acessar a cada uma de suas posições.

Em outras palavras, um array multidimensional é como um “contêiner” que guardará mais valores para cada posição, ou seja, como se os elementos do array fossem por sua vez outros arrays.

Outra ideia que temos é que matrizes são arrays nos quais algumas de suas posições podem conter outros arrays de forma recursiva. Um array multidimensional pode ser criado pela função array():

Na figura abaixo temos a representação de um array com duas dimensões.

					Colunas	
					0 1 2 3 4	
Linhas	0	0.0	0.1	0.2	0.3	0.4
	1	1.0	1.1	1.2	1.3	1.4
	2	2.0	2.1	2.2	2.3	2.4
	3	3.0	3.1	3.2	3.3	3.4
	4	4.0	4.1	4.2	4.3	4.4

Uma diferença importante de um array comum para um multidimensional é a quantidades de chaves (índices), onde cada um dos índices representa uma dimensão.

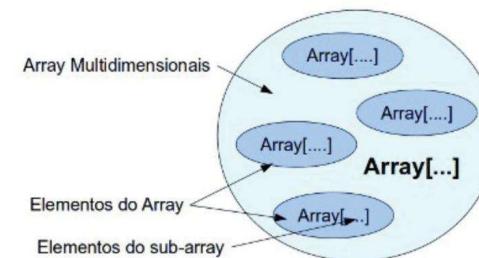
Código:

```
289<script type="text/javascript">
290    var matriz = new Array(
291        new Array("0.0" , "0.1" , "0.2" , "0.3" ) ,
292        new Array("1.0" , "1.1" , "1.2" , "1.3" ) ,
293        new Array("2.0" , "2.1" , "2.2" , "2.3" ) ,
294        new Array("3.0" , "3.1" , "3.2" , "3.3" ) );
295 </script>
```

Outra forma de iniciar o array:

```
301<script type="text/javascript">
302    var matriz = new Array(4,4);
303    matriz[0] = new Array("0.0" , "0.1" , "0.2" , "0.3" );
304    matriz[1] = new Array("1.0" , "1.1" , "1.2" , "1.3" );
305    matriz[2] = new Array("2.0" , "2.1" , "2.2" , "2.3" );
306    matriz[3] = new Array("3.0" , "3.1" , "3.2" , "3.3" );
307 </script>
```

Observe que temos uma chave para representar a linha e outra para representar a coluna, assim, determinando uma matriz 4x4. Podemos ver também que inicializamos um array dentro do outro. Cada sub- array é uma linha, e cada elemento do array maior representa as colunas.



Para acessarmos o valor de um array multidimensional, basta colocar as duas ou mais chaves da posição que queremos acessar. É muito semelhante ao array de uma única dimensão.

Observe o acesso aos exemplos anteriores:

nme\_do\_array[chave\_1][chave\_2]...[chave\_n];

**Exemplo :**

```
310<script type="text/javascript">
311    var matriz = new Array(4,4);
312    matriz[0] = new Array("0.0" , "0.1" , "0.2" , "0.3" );
313    matriz[1] = new Array("1.0" , "1.1" , "1.2" , "1.3" );
314    matriz[2] = new Array("2.0" , "2.1" , "2.2" , "2.3" );
315    matriz[3] = new Array("3.0" , "3.1" , "3.2" , "3.3" );
316
317    //acessando array linha 1 coluna 2
318    document.write(matriz[1][2]); //Resultado: 1.2
319 </script>
```

Dessa forma podemos acessar o elemento “1.2” que está guardado na posição linha 1 coluna 2, lembrando que o primeiro elemento de um array é 0. Abaixo, um exemplo que acessa todos os valores do array e imprime com quebra de linha:

**Código:**

```

324<script type="text/javascript">
325  var matriz = new Array(
326    new Array("0.0", "0.1", "0.2", "0.3"),
327    new Array("1.0", "1.1", "1.2", "1.3"),
328    new Array("2.0", "2.1", "2.2", "2.3"),
329    new Array("3.0", "3.1", "3.2", "3.3"));
330 //percorrendo matriz
331 for(i=0 ; i<matriz.length;i++){
332   for(j=0 ; j<matriz[i].length;j++){
333     //retirando a ultima vírgula da linha na hora da impressão
334     if(j != matriz[i].length-1){
335       document.write(matriz[i][j]+", ");
336     }
337     else{
338       document.write(matriz[i][j]);
339     }
340   }
341   document.write("<br>");
342 }
343 </script>

```

**Resultado:**

```

0.0, 0.1, 0.2, 0.3
1.0, 1.1, 1.2, 1.3
2.0, 2.1, 2.2, 2.3
3.0, 3.1, 3.2, 3.3

```

**Explicando o código:**

Linha 325 à 329 – criamos um array de duas dimensões.

Linha 331 – temos um for que irá percorrer as linhas de nossa matriz. Esse for tem uma variável i iniciando com 0 e será incrementado de um em um enquanto i for menor que o número de linhas da matriz

Linha 332 – temos um for que irá percorrer as colunas de nossa matriz. Esse for tem uma variável j iniciando com 0 e será incrementado de um em um enquanto j for menor que o número de colunas da matriz

Linha 334 – temos um if que irá verificar se não estamos na última coluna da linha que está sendo impressa. Se não estiver na última coluna irá imprimir o valor da coluna concatenado com uma “, ”.

Linha 337 – caso a condição da linha 334 não seja verdade, é porque iremos imprimir a última coluna da linha que está sendo impressa, então irá imprimir somente o valor da coluna, sem a vírgula.

Linha 341 – temos uma quebra de linha, que será executada todas as vezes que terminarmos de imprimir todos os valores de uma linha, ou seja quando o for mais interno for finalizado

**7.6 Funções com Arrays**

O objeto Array possui uma propriedade chamada length. Esta propriedade mostra quantos elementos o vetor possui.

**Código:**

```

<script type="text/javascript">
  vetor = new Array("João", "Roberto", "José");
  document.write(
    "<p>Este vetor possui "+vetor.length+" elementos. Que são: </p>");
  document.write("<ul>");
  for (indice = 0; indice < vetor.length; indice++) {
    document.write ("<li>"+ vetor [indice]+ "</li>")
  };
  document.write("</ul>");
</script>

```

Saída:

Este vetor possui 3 elementos. Que são:  
João  
Roberto José

**7.7 Junção e Concatenação de Arrays (Vetores)**

Como vimos anteriormente, os vetores em JavaScript são constituídos pelo objeto Array.

Há momentos que queremos unir um ou mais vetores. JavaScript nos possibilita fazer isso resultando duas formas diferentes.

**7.8 Método join()**

O vetor construído com o objeto Array de JavaScript pode ter a junção de seus elementos. A junção consiste em criar uma string única usando separadores.

O método **join()** nos possibilita a junção de forma simples, precisa e correta em uma string.

Por exemplo, vamos imaginar que temos um vetor que possui os dados de uma data qualquer em seus índices e que queremos mostrar a saída dessa data inteira. Há duas formas que podemos fazer isso e são complicadas.

Uma delas é criar um loop entre os dois primeiros índices e depois mostrar apenas o último índice.

```
<script type="text/javascript">
  var data = new Array(3);
  data[0] = 27;
  data[1] = 3;
  data[2] = 2009;

  for (i = 0; i<data.length-1; i++) {
    document.write (data[i] + "/");
  }
  document.write (data[2]);
</script>
```

A outra forma, apesar de mais simples, ainda não é a mais indicada.

```
<script type="text/javascript">
  var data = new Array(3);
  data[0] = 27;
  data[1] = 3;
  data[2] = 2009;
  document.write (data.join("/"));
</script>
```

Vale lembrar que o método join não modifica o vetor original, mas é possível guardar seu resultado em uma variável. Ex.: dataCompleta = data.join("/") .

### 7.8.1 Método concat()

O método **concat()** consiste em unir um ou mais arrays (vetores). Este método usa como argumento um objeto do tipo array. Se desejarmos unir mais de um vetor, cada objeto array do argumento deve vir separado por vírgula (,).

**Código:**

```
<script type="text/javascript">
  var alunosAno1 = new Array ("André","Sansão","Raquel");
  var alunosAno2 = new Array ("Marcos","Valdenia","Fabiana");
  var alunosAno3 = new Array ("Thiago","Elinardy","Krystian");
  var alunosAno4 = new Array ("Fabricio","Patrício","Atila");
  var todosAlunos = alunosAno1.concat(alunosAno2,alunosAno3,alunosAno4);
  document.write ("Esta escola tem " + todosAlunos.length +
    " alunos no projeto e-Jovem.<br>" + "Eles se chamam:<br>" + todosAlunos);
</script>
```

Aqui, o resultado obtido na variável todosAlunos é o nome contido nas quatro variáveis alunosAno.

**Saída:**

Esta escola tem 12 alunos no projeto e-Jovem. Eles se chamam:

André,Sansão,Raquel,Marcos,Valdenia,Fabiana,Thiago,Elinardy,Krystian,Fabricio,Patrício,Atila

Assim como no método join(), concat() também não altera o conteúdo original dos vetores.

### 7.8.2 Método push()

Além da forma algorítmica de inserir um elemento no final de um array, o próprio objeto Array possui um método que faz isso por nós.

O método push() insere um ou mais elementos no final de um array. A sintaxe é: push (elemento1, elemento2, elemento3, ...)

Dessa forma, não precisamos nos preocupar com o tamanho e nem com o último índice do array.

**Código:**

```
<script type="text/javascript">
  vetor = new Array ("Cristiane");
  document.write ("Este array possui "+vetor.length+" elementos.<br>");
  vetor.push ("Nogueira");
  document.write("Agora possui "+vetor.length+" elementos.<br>");
  vetor.push ("Wallison","Ledruwick","Crispiano","Mateus");
  document.write ("Este vetor possui "+vetor.length+
    " elementos.<br>"+"Que são: "+vetor.join(", "));
</script>
```

**Saída:**

Este array possui 1 elementos.  
Agora possui 2 elementos.  
Este vetor possui 6 elementos.  
Que são: Cristiane, Nogueira, Wallison, Ledruwick, Crispiano, Mateus

### 7.8.3 Método unshift()

Outra forma de inserir elementos é utilizar o método unshift(). O método unshift() insere um ou mais elementos no início de um array.

A sintaxe é: unshift (elemento1, elemento2, elemento3, ...)

Um detalhe importante é o fato de que este método possivelmente não funcione corretamente em versões

anteriores ao Internet Explorer 6.

#### Código:

```
<script type="text/javascript">
    vetor = new Array ("5","6","7","8","9","10","J","Q","K");
    document.write ("Este baralho tem apenas as cartas "+vetor.join(", ")+".
        Vamos completá-lo com as cartas iniciais. <br>");
    vetor.unshift ("A","1","2","3","4");
    document.write("Agora temos o baralho completo: "+vetor.join(", "));
</script>
```

#### Saída:

Este baralho tem apenas as cartas 5, 6, 7, 8, 9, 10, J, Q, K. Vamos completá-lo com as cartas iniciais.  
Agora temos o baralho completo: A, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K

### 7.8.4 Método pop()

O método **pop()** remove um elemento do final de um array e retorna o conteúdo do índice removido.

Não precisamos nos preocupar com o tamanho e nem com o último índice do array, pois **pop()** fará todo o trabalho sozinho.

#### Código:

```
<script type="text/javascript">
    vetor = new Array ("Marcelo","Patricia","Joelma","Diana");
    document.write ("Este array possui "+vetor.length+" elementos.<br>");
    document.write ("Vamos remover "+vetor.pop()+" da lista.<br>");
    document.write ("Agora, temos "+vetor.length+" elementos. <br>"+
        "Que são: "+vetor.join(", "));
</script>
```

#### Saída:

Este array possui 4 elementos.  
Vamos remover Diana da lista.  
Agora, temos 3 elementos.  
Que são: Marcelo, Patricia, Joelma

### 7.8.5 Método shift()

Outra forma de remover elementos é utilizar o método **shift()**. O método **shift()**remove um elemento do início de um array e retorna o conteúdo do índice removido.

#### Código:

```
<script type="text/javascript">
    vetor = new Array ("Marcelo","Patricia","Joelma","Diana");
    document.write ("Este array possui "+vetor.length+" elementos.<br>");
    document.write ("Vamos remover "+vetor.shift()+" da lista.<br>");
    document.write ("Agora, temos "+vetor.length+" elementos. <br>"+
        "Que são: "+vetor.join(", "));
</script>
```

#### Saída:

Este array possui 4 elementos.  
Vamos remover Marcelo da lista.  
Agora, temos 3 elementos.  
Que são: Patricia, Joelma, Diana

### 7.8.6 Método reverse()

O método **reverse** do objeto array serve para inverter a ordem dos elementos de um array. Dessa forma podemos usar apenas uma forma de organização e depois inverter o array para conseguir a ordem desejada.

#### Código:

```
<script type="text/javascript">
    var vetor = Array (1,2,3,4,5);
    document.write ("Array inicial: " + vetor);
    document.write ("<br> Array invertido: " + vetor.reverse());
</script>
```

#### Saída:

Array inicial:1,2,3,4,5  
Array invertido: 5,4,3,2,1

### 7.8.6 Método sort()

O método **sort** é o que realmente faz a organização do array. Porém, **sort()** apenas organiza o array de forma alfabética, se quisermos organizar o array de forma numérica devemos criar uma função para isso.

Abaixo está um exemplo de uma organização simples feita em ordem alfabética.

#### Código:

```
<script type="text/javascript">
    vetor = new Array ("Elinny","André","Adriano","Elisabete");
    document.write (vetor.sort());
</script>
```

Veja que *vetor* é organizado corretamente de forma alfabética.

**Saída:**

Adriano,André,Elinny,Elisabete

Agora, podemos perceber que o resultado da organização de números não é dado de forma satisfatória.

**Código:**

```
<script type="text/javascript">
    vetor = new Array (3000,20,100,4);
    document.write (vetor.sort());
</script>
```

**Saída:**

100,20,3000,4

### Exercícios Resolvido

- Quais serão as letras que serão geradas a partir desse código?

```
347<script type="text/javascript">
348    var valor = 2;
349    var indice = valor;
350    var letras = new Array('d','b','e','a','i','o','r');
351
352    document.write(letras[indice++]);
353    document.write(letras[indice]);
354    document.write(letras[indice-2]);
355    document.write(letras[0]);
356    document.write(letras[valor+1]);
357 </script>
```

### Entendendo o algoritmo

- No escopo temos uma variável chamada “valor” = 2 e “índice” que recebe o conteúdo que está em valor que é 2 também.
- Temos também um array que irá receber um conjunto descrito, lembrando que o índice inicial de um vetor é zero até a sua dimensão menos um. Nesse caso temos 7 elementos, então os índices para o meu vetor são de zero até 6.

- Para cada impressão, temos uma letra seguido de uma concatenação com um espaço em branco, então podemos chegar a conclusão que irá ser impresso um conjunto de letras com e espaços.
- O primeiro “document.write” irá imprimir o valor de “índice” = 2 que é representado pela letra “e”, em seguida índice será incrementado passando a ser igual a 3.
- O segundo “document.write” irá imprimir o valor de índice = 3 que é representado pela letra “a”.
- O terceiro “document.write” irá imprimir o valor de (índice - 2) = ( 3-2 ) = 1 que é representado pela letra “b”
- O quarto “document.write” irá imprimir o índice zero do meu array representado pela letra “d”
- O quinto “document.write” irá imprimir o índice (valor +1) = (2 + 1) = 3 representando pela letra “a”
- Por fim podemos identificar a mensagem no JAVASCRIPT: e a b d a.

### Exercícios Propostos:

O que é um array, e qual a sua principal finalidade? Declare um array chamado “nomes” com 8 posições, e grave nomes de pessoas que você conhece em cada uma delas. Após criar o array responda as questões 2, 4, 5, 10, 11:

Utilizado o array responda.

- Qual nome é impresso no navegador se colocarmos o código:

document.write(nomes[3]);

- Quais nomes aparecerá se adicionarmos os seguintes códigos:

for(i= 6; i>1 ; i-){ document.write(nomes[i]); }

- O que acontece se chamarmos uma posição que não existe, exemplo: nomes[15];

O que é um array associativo, de exemplos:

Usando o comando *for...in*, crie uma interação onde todos os nomes possa ser impresso na tela.

Utilizando o mesmo código, altere alguns nomes do array.

O que é um array multidimensional?

Crie um array “palavras” multidimensional 5x3 com os valores da tabela abaixo, e responda as questões 7,8,9:

“oi”	“tudo”	“estar”
“você”	“vai”	“?”
“com”	“dia”	“!”
“ ”	“bem”	“sim”
“casa”	“hoje”	“em”

Crie um código JAVASCRIPT onde com os valores do array possa ser impresso na tela com a frase “oi, tudo bem com você?”.

Utilizando as posições da sequência [1][0],[1][1],[0][2],[4][2],[4][0],[4][1],[1][2] do array palavras, qual frase podemos formular? Utilize a função document.write() para mostrar na tela do navegador.

Construa um código JAVASCRIPT para mostra uma resposta para a pergunta da questão 7. (Use o comando document.write para imprimir na tela).

Utilizando a função sort(), imprima em ordem alfabética os nomes do array “nomes”.

Use o comando unshift() para adicionar mais dois nomes no array.

Crie um documento HTML com um código JavaScript que armazene 10 números inteiros em um array. Preencha todas as posições do array com valores sequenciais e em seguida imprima-os na tela. Em seguida, escolha duas posições aleatoriamente e troque os valores de uma posição pelo da outra. Repita essa operação 10 vezes. Ao final, imprima o array novamente.

Crie um documento HTML com um código JavaScript que armazene 10 números inteiros em um array. Preencha todas as posições com valores aleatórios e em seguida imprima-os. Ordene do menor valor para o maior e imprima novamente.

## 8.0 Manipulação de Funções

Quando queremos um código funcional para determinado fim, com por exemplo fazer um cálculo ou alguma interação dentro do JavaScript, usamos o que chamamos de função. As funções são um pedaço de código com o objetivo específico, encapsulado sob uma estrutura única que recebe um conjunto de parâmetros e retorna ou não um determinado dado. Uma função é declarada uma única vez, mas pode ser utilizada diversas vezes. É uma das estruturas mais básicas para prover reusabilidade ou reaproveitamento de código, deixando as funcionalidades mais legíveis.

### 8.1 Declarando uma Função.

Declaramos uma função, com o uso do operador function seguido do nome que devemos obrigatoriamente atribuir, sem espaços em branco e iniciando sempre com uma letra. Temos na mesma linha de código a declaração ou não dos argumentos pelo par de parênteses “()”. Caso exista mais de um parâmetro, usamos vírgula(,) para fazer as separações. Logo após encapsulamos o código pertencente a função por meio das chaves ({}). No final, temos o retorno com o uso da cláusula return para retornar o resultado da função que pode ser um tipo inteiro, array, string, ponto flutuante etc. A declaração de um retorno não é obrigatório. Observe a sintaxe:

```
function nome_da_função( argumento_1, argumento_2, argumento_n )
{
    comandos; return $valor;
}
```

Observe um exemplo onde criamos uma função para calcular o índice de massa corporal de uma pessoa (IMC), onde recebe como parâmetro dois argumentos. Um é a altura representada pela variável \$altura e o outro é o peso representada pela variável \$peso. Passamos como parâmetros para essa função o peso = 62 e a altura = 1.75. Observe:

```
<script type="text/javascript">
//Declarando a função
function calculo_IMC(peso, altura){
    return peso/(altura*altura);
}
//Chamando a função
document.write(calculo_IMC(62,1.75));
</script>
```

**Exemplo:**

**Resultado:** 20.244897959183675

Nesse exemplo temos a declaração e logo após a chamada da função, onde é nesse momento que passamos os dois parâmetros na ordem que foi declarada na função. Lembrando que essa ordem é obrigatória. Observe mais um exemplo, onde a função declarada, porém não possui a cláusula **return**.

```
<script type="text/javascript">
    //Declarando a função
    function imprime(texto){
        document.write("<h1>" + texto + "</h1>");
    }
    //Chamando a função
    imprime("Testando a função.");
</script>
```

## 8.2 Escopo de Variáveis em Funções

Um conceito importante em programação são os tipos de declarações de variáveis, onde sua visibilidade vai depender de onde ela é declarada. O acesso a essas variáveis pode ser definido da seguinte forma:

**Variáveis locais** < São aquelas declaradas dentro de uma função e não tem visibilidade fora dela.

**Variáveis Globais** < São variáveis declaradas fora do escopo de uma função, porém tem visibilidade (pode ser acessada) ao contexto de uma função sem passá-la como parâmetro. Para isso declaramos a variável no escopo fora da função e fazemos a sua chamada dentro da função.

## 8.3 Valor de Retorno

Toda função pode opcionalmente retornar um valor, ou simplesmente executar os comandos e não retornar valor algum. Não é possível que uma função retorne mais de um valor, mas é permitido fazer com que uma função retorne um valor composto, como listas ou array's. As operações aritméticas podem ser feitas de forma direta no retorno. Observe um exemplo onde temos uma operação direta:

```
<script type="text/javascript">
    function somar(n1,n2){
        return n1+n2;
    }
    function texto () {
        return "O resultado é: ";
    }
    document.write(texto() + somar(115,23));
    //Resultado: 138
</script>
```

Também podemos determinar mais de um retorno desde que eles não sejam acessados ao mesmo tempo, observe o exemplo:

```
<script type="text/javascript">
    function teste(caso){
        switch(caso){
            case 1:
                return "opção 1";
            case 2:
                return "opção 2";
            case 3:
                return "opção 3";
            default:
                return null;
        }
    }
    alert(teste(2));
</script>
```

Esse código mostra de forma clara que não existe a possibilidade de retornarmos mais de um **return**, caso isso ocorresse, teríamos um erro, ou não funcionamento da função.

## 8.4 Recursão.

Função recursiva é uma definição usada tanto na programação quanto na matemática, onde, significa que uma função “faz a chamada” de si mesma na sua execução. Um exemplo é o cálculo do fatorial de um número. Observe:

```
<script type="text/javascript">
    function fatorial(numero){
        if (numero === 1) {
            return numero;
        }else{
            return numero * fatorial(numero-1);
        }
    }
    alert(fatorial(5)); //Resultado: 120
</script>
```

**Fatorial de 5:**  $5! = 5 \cdot 4! \cdot 4! = 4 \cdot 3! \cdot 3! = 3 \cdot 2! \cdot 2! = 2 \cdot 1! \cdot 2! = 1 \cdot 1! \cdot 2! = 120$ .

### Exercícios Propostos

Diga com suas palavras uma definição para função, e como podemos declará-la em PHP.

Qual a diferença de variáveis globais para variáveis locais e como podemos defini-las em PHP?

Quais as funções que podemos usar para criarmos uma função onde seus parâmetros são passados por argumentos variáveis?

O que é um valor de retorno e qual o comando usado quando queremos retornar algo dentro de uma função?

O que é recursão?

---

Crie uma função que determine se um numero é par ou ímpar. E faça uma chamada dessa função imprimindo o resultado.

Crie uma função que calcule a factorial de um número.

Crie uma função para determina se um numero é primo ou não. Numero primo é aquele que possui dois divisores, 1 e ele mesmo. Criem um laço de repetição e use estrutura de controle.