

# Apostila JavaScript

## Unidade 2 – Utilizando Funções

### 2.1 O que é Função

Uma função é um grupo de linha(s) de código de programação destinado uma tarefa bem específica e que podemos de necessário, utilizar várias vezes. A utilização de funções melhora bastante a leitura do script.

### 2.2 Utilizando Funções no JavaScript

Como já podemos perceber, o Javascript possui inúmeros recursos e funções disponíveis para tornar as páginas mais interativas possíveis. Além de toda a estrutura de programação já apresentada, o javascript ainda permite ao desenvolvedor utilizar funções próprias do Javascript e criar suas próprias funções.

### 2.3 As funções intrínsecas do JavaScript

O Javascript possui uma série de funções intrínsecas que podem ser chamadas em qualquer ponto do código JavaScript. As funções intrínsecas mais comuns são apresentadas a seguir.

#### 2.3.1 A função parseInt() e parseFloat()

As funções **parseInt** e **parseFloat** são utilizadas para **converter** valor do formato string (texto) para o formato numérico. A função **parseInt** é utilizada para converter valores inteiros. A função **parseFloat** é utilizada para converter valores fracionados. Estas funções são extremamente úteis quando se faz necessária a conversão de um valor textual, informado pelo usuário ou não, para um tipo número. A partir da conversão é possível utilizar os valores para cálculos numéricos. Podemos ver exemplos destas duas funções no script a seguir:

```
<script language="javascript" type="text/javascript">
  var nContador;
  var sValor1, sValor2;
  var sResultado, nResultado;
  sValor1 = "10"; //Valor String
  sValor2 = "1.5"; // Valor Fracionado
  sResultado = sValor1 + sValor2;
  nResultado = parseInt(sValor1) + parseFloat(sValor2);
  // Concatenação
  alert(sResultado + " " + nResultado);
  // Soma numérica
  alert(nResultado);
</script>
```

#### 2.3.2 Função isNaN()

A função **isNaN** (Is Not a Number) é utilizada para verificar se uma variável contém um valor especial NaN. O valor representa uma conversão numérica mal sucedida (NaN = Not a

Numeric). Quando tentamos converter uma string (Ex.: ABC) para um valor numérico utilizando as funções **parseInt** e **parseFloat**, o resultado da função é um valor NaN.

```
<script language="javascript" type="text/javascript">
    var sTelefone = "1234XDFRT";
    //validar telefone (verificação se contem apenas números)
    if (isNaN(sTelefone)){
        alert ("0 seu telefone " + sTelefone + " deve conter
            apenas números!");
    }
</script>
```

## 2.4 Funções criadas pelo usuário

Além das funções disponíveis, por padrão no Javascript, a linguagem nos permite criar nossas próprias funções, chamadas funções definidas pelo usuário. Para criar funções devemos utilizar a cláusula **function**, que é utilizada para criar as funções no Javascript. A sintaxe para a criação das funções no JavaScript é:

```
function <Nome da Função> ([param1], [param2], ... , [paramn])
{
    ...
}
```

Como podemos ver a função pode conter diversos **parâmetros** de entrada que podem ser utilizados para cálculos ou algum processamento especial dentro do corpo da função. Para definir os parâmetros, basta informar os nomes das variáveis que irão representar os parâmetros na função. Podemos ver um exemplo de função no código abaixo:

```
<script language="javascript" type="text/javascript">
    function Soma(nNumero1, nNumero2)
    {
        var nSoma;
        nSoma = nNumero1+nNumero2;
        alert("Soma= " + nSoma);
    }
    Soma(10,5);
</script>
```

As funções do Javascript ainda permitem que o **usuário retorne valores**. Para retornar um valor em uma função, basta criar a função da mesma forma como apresentado no exemplo anterior, porém com a utilização do comando **return**. O comando **return** serve para interromper o processamento de uma função ou um bloco de script, com a opção de retornar um valor no caso de estar dentro de uma função. Podemos ver um exemplo de utilização do comando **return** a seguir.

```
<script language="javascript" type="text/javascript">
    function Soma(nNumero1, nNumero2)
    {
        return nNumero1+nNumero2;
    }
    alert("Soma= " + Soma(10,5);
</script>
```

OBS: quando utiliza-se funções deve-se ter atenção com a questão de **variáveis globais e locais**.

## Unidade 3 – Expressões

### 3.1 Expressões Condicionais – estruturas de teste

Para uma utilização eficaz dos operadores disponíveis no Javascript, é necessário entender como funcionam as estruturas de teste utilizadas pela linguagem. Dentro do Javascript possuímos o `if else`.

#### 3.1.1 `if... else`

No comando `if`, é necessário informar **dentro de parênteses a expressão** a ser testada, e caso ela seja verdadeira, o comando ou bloco de comandos subsequente ao comando `if` será executado. Este comando ainda possui uma cláusula **`else` opcional**, que pode conter uma expressão ou conjunto de expressões a serem executadas caso a condição testada no comando `if` não seja verdadeira.

A seguir podemos verificar um exemplo do comando `if`:

##### **Exemplo 1 – *if* simples:**

```
<script language="javascript1.1" type="text/javascript">
var sTelefone;
sTelefone = prompt("Informe o número do telefone","");
//validar telefone (verificação se contem apenas números)
if (isNaN(sTelefone))
    alert ("O seu telefone deve conter apenas números!");
</script>
```

##### **Exemplo 2 – *if else*:**

```
<script language="javascript1.1" type="text/javascript">
var nNum;
nNum = parseInt(prompt("Informe um número inteiro:", ""));
if(nNum > 0)
    alert("Número positivo");
else
    alert("Número negativo");
</script>
```

Para aqueles que gostam da escrita enxuta, também há:

```
(expressão)?instruçãoA:instruçãoB;
```

Se a expressão entre parêntese é verdadeira, a instrução A será executada. Se a expressão entre parêntese é falsa, é a instrução B será executada.

```
<script language="javascript1.1" type="text/javascript">
var nNum;
nNum = parseInt(prompt("Informe um número inteiro:", ""));
alert((nNum>0)? "Número positivo":"Número negativo");
</script>
```

### Exemplo 3 – if else encadeado:

```
<script language="javascript" type="text/javascript">
  var nNum;
  nNum = parseInt(prompt("Informe um número inteiro:", ""));
  if(nNum == 0)
    alert("Número neutro");
  else if(nNum > 0)
    alert("Número positivo");
  else
    alert("Número negativo");
</script>
```

Se o comando *if* possuir mais de uma instrução, é obrigatório a utilização dos delimitadores de “{ }”.

## 3.2 Expressões Seletoras

### 3.2.1 Switch

Além do comando *if*, o comando *switch* pode ser utilizado para realizar testes lógicos na linguagem Javascript. O comando *switch* é um comando de **teste de seleção múltipla**, ou seja, não testa a expressão lógica apenas por um valor específico, mas sim por uma série de valores informados no código de script. O comando *script* requer um bloco de comando(s) inserido(s) logo após a expressão lógica, que irá conter os comandos executados no caso de cada uma das expressões. Ao final de cada conjunto de instruções de uma condição *switch*, é necessário informar o **comando break**, para que o javascript não execute as condições posteriores ao teste, conforme o exemplo a seguir:

```
<script language="javascript1.1" type="text/javascript">
var farol;
farol = prompt("Informe a cor do semáforo", "");
switch (farol) {
  case "vermelho":
    alert("Pare");
    break;
  case "amarelo":
    alert("Atenção");
    break;
  case "verde":
    alert("Prossiga");
    break;
  default:
    alert("Cor ilegal");
}
```

## 3.3 Expressões de Loop - Estruturas de repetição

### 3.3.1 While

O comando *while* é utilizado para repetir um determinado conjunto de instruções de acordo com uma expressão lógica definida no código de script.

```
while (condição verdadeira){  
  continuar a fazer alguma coisa  
}
```

Enquanto que a **condição é verdadeira**, o Javascript continua a executar as instruções entre as chaves. Uma vez que a condição não é mais verdadeira, o ciclo interrompe-se e continua-se o script. Podemos ver um exemplo de utilização do comando while no código abaixo:

```
<script language="javascript1.1" type="text/javascript">  
var sNome, nCont;  
nCont = 1;  
while(nCont <=5)  
{  
  sNome = prompt("Informe o nome: ", "");  
  //Incrementa o contador  
  nCont++;  
}
```

### 3.3.2 do while

A estrutura de repetição do while funciona da mesma forma que a estrutura while, porém o teste de continuidade do loop é realizado após o bloco de código. Isso significa que o código será executado **ao menos uma vez**.

```
var text = ""  
var i = 0;  
do {  
  text = "<br>The number is " + i;  
  document.write(text);  
}  
while (i < 5);
```

### while X do .. while



### 3.4.1 For

Além do comando while, temos o comando for, que pode ser utilizado para repetir um determinado conjunto de instruções de acordo com um índice número especificado no próprio comando for. No comando for é necessário informar a variável que servirá como índice, a condição

para execução do for, e o incremento utilizado para execução da estrutura de repetição.

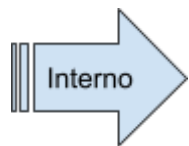
```
for(valor inicial ; condição ; progressão)

<script language="javascript1.1" type="text/javascript">
var sNome, nCont;
for(nCont=1;nCont<=5;nCont++)
{
    sNome = prompt("Informe o nome: ","");
}
</script>
```

### 3.4.2 Loop Aninhados

Aninhar um loop consiste colocar esse loop dentro de outro. O aninhamento de loops é necessário para fazer determinados processamentos um pouco mais complexos dos que os que vimos nos exemplos anteriores.

Um loop aninhado tem a estrutura como a que segue abaixo. Vamos explicá-lo através destas linhas:



```
let e,n;
document.write("2 Estudantes 3 Notas" + "<br>");
for (e = 0; e < 2; e++) {
    document.write("Estudante " +e + ":");
    for (n = 0; n < 3; n++) {
        document.write(Math.random()*10 +', ');
    }
    document.write("<br/>");
}
```

O loop que está aninhado (mais para dentro) é o que mais se executa, neste exemplo, para cada repetição do loop mais externo, o loop aninhado executará por completo uma vez, ou seja, fará suas 3 repetições.

### 3.4 O Objeto Array

O Objeto Array (também conhecido como matriz) é a representação de um conjunto de valores dentro do Javascript. Para criarmos um array dentro do Javascript, utilizamos a mesma sintaxe que utilizamos até o momento com as datas, através do **comando new**. Deve-se lembrar que a primeira posição **de um array é 0**, então um array de tamanho 6, começa na posição 0 e acaba na posição 5. A seguir vemos uma série de exemplos de criação de arrays.

```
<script language="javascript1.1" type="text/javascript">
var sNomes = new Array();
var sValores = new Array(5) // Array de 5 posições;
</script>
```

Como mostrado no exemplo anterior, os array podem ser criados com um tamanho inicial ou não. Caso não seja informado o tamanho inicial, **podemos aumentar seu tamanho à medida que criamos os elementos dentro do array**. Para criar e acessar elementos em um array em Javascript devemos utilizar colchetes [ ] para informar qual o início do array que queremos gravar ou ler. Caso o início não exista e não foi atribuído um tamanho para ele, o array será redimensionado automaticamente. Caso contrário, o script exibirá um erro.

| PROPRIEDADE | DESCRIÇÃO   |
|-------------|---|
| length      | Devolve o número de elementos (tamanho) do array. |

Caso o array esteja sendo preenchido de forma dinâmica, sem informar um número de elementos, podemos a qualquer momento consultar a propriedade length para verificar quantos elementos estão contidos dentro do array.

Além da armazenagem de objetos, os array podem ser utilizados para outras finalidades como **ordenação** de dados. Para isto devemos utilizar os métodos disponíveis dentro do array, como o método **sort** para ordenar por ordem alfabética e o método **reverse** para ordenar por ordem decrescente.

Um exemplo de utilização de um array para ordenação pode ser visto abaixo:

```
<script language="javascript1.1" type="text/javascript">
var sNomes    = new Array();

sNomes[0] = "Carlos";
sNomes[1] = "Bruno";
sNomes[2] = "Mauricio";
sNomes[3] = "José";
sNomes[4] = "André";

sNomes.sort();

var nCount;
alert("Ordem Crescente");
for(nCount=0;nCount<sNomes.length;nCount++)
{
    alert(sNomes[nCount]);
}

sNomes.reverse();
alert("Ordem Decrescente");
for(nCount=0;nCount<sNomes.length;nCount++)
{
    alert(sNomes[nCount]);
}
</script>
```



**Para classificar array numéricos, utilizar a seguinte expressão:**

```
var ar= new Array();  
ar[0]=(10);  
ar[1]=(8);  
ar[2]=(5);  
ar.sort(function(a, b){return a-b}); //ordem crescente  
ar.sort(function(a, b){return b-a}); //ordem decrescente  
for (i = 0; i < 3; i++) {  
    document.write(ar[i] + ",");  
}
```

## **Lista de Exercício 2**