JavaScript

- ✓ É uma linguagem de script criada pela Netscape.
- ✓ Linguagem de script significa que é executada no interior de programas ou de outras linguagens de programação.
- ✓ Por ser uma linguagem de script, também é considerada uma linguagem interpretada (não é compilada), pois seu código fonte é executado por um interpretador e, na sequência, pelo sistema operacional ou processador.
- ✓ As aplicações desenvolvidas em JavaScript podem rodar em browsers (navegadores) ou em servidores.
- ✓ A linguagem de programação JavaScript é utilizada principalmente para aumentar a interatividade entre as páginas web e os usuários.
- ✓ ECMA (European Computer Manufacturers Association) é uma associação internacional de padronização de informações e sistemas de comunicação. A versão padronizada do JavaScript, chamada ECMAScript, se comporta do mesmo jeito em todas as aplicações que suportam esse padrão.

- **Estrutura da Linguagem:**

  - ✓ O código em JavaScript deve ser escrito entre as tags <script> e </script>. Antigamente, utilizava-se também o atributo *type*: <script type="text/javascript">. Este atributo não é obrigatório, já que o JavaScript é a linguagem padrão de scripts no HTML.
  - ✓ Os comandos são escritos com letras minúsculas.
  - ✓ O JavaScript diferencia caracteres maiúsculos e minúsculos.
  - ✓ Comentários: `/* ... */` ou `//` (para uma linha inteira)
  - ✓ É colocado um sinal de ponto e vírgula (;) para indicar o final de uma linha de comando.
  - ✓ Em estruturas de decisão e repetição ("if", "for" e "while") os blocos de instruções que contenham mais de uma linha de código devem estar entre "{ }" (chaves). A declaração de funções também deve ter todo o código delimitado por chaves.
  - ✓ Os scripts são incluídos em páginas HTML de duas formas:
    a. Interno: coloca-se as instruções entre as tags <script> e </script>. A tag <script> pode ser inserida dentro das tags <head>, <body> ou em ambas. Como regra geral, coloca-se dentro da tag <body> as instruções do script que devem ser executadas na hora em que a página for acessada, caracterizando uma rotina principal e dentro da tag <head> as partes do script que serão executadas na forma de função, sendo considerada uma rotina secundária (sub-rotina).
    Colocar scripts na parte inferior do elemento <body> contribui para a velocidade de carregamento da página.
    Exemplo:

```
<body>
    <script>
        /* Script de Boas Vindas */
        var NOME;
        NOME = prompt('Informe o seu nome:', 'Digite aqui');
        document.write('<h1> Olá ' + NOME.toUpperCase() + '!</h1>');
        document.write('<h2>Seja Bem-Vindo!</h2>');
    </script>
</body>
```

    b. Externo: Insere-se o código JavaScript dentro de um arquivo com extensão .js e chama este arquivo no atributo src da tag <script>.
    Exemplo:

```
<script src="arquivo_externo.js"></script>
```

✓ Em JavaScript, o mais comum é que os comandos sejam executados quando o usuário realizar alguma ação, como por exemplo, clicar em um elemento. Para isso é necessário utilizar dois recursos do JavaScript: **Funções** e **Eventos**
Ao criarmos uma **função**, a execução do código só será realizada quando a função for chamada.
Um **evento** permite determinar o momento em que funções e/ou comandos devem ser executados.
Também é possível chamar uma função sem a necessidade de um evento. Para isso, deve-se escrever o nome da função abrindo e fechando parênteses.

Existem diversos **eventos** que podem ser utilizados para disparar funções. Alguns exemplos:

- **onblur**: quando um elemento perder o foco
- **onchange**: quando um input, select ou textarea tiver o seu valor alterado
- **onclick**: ao clicar com o mouse
- **ondblclick**: ao clicar duas vezes com o mouse
- **onfocus**: quando um elemento ganhar o foco
- **onkeypress**: ao pressionar e soltar uma tecla
- **onload**: quando o elemento é carregado (página, imagem, script, CSS, entre outros)
- **onmousemove**: ao mexer o mouse
- **onmouseover**: quando o ponteiro do mouse passa a estar sobre um elemento
- **onscroll**: quando a barra de rolagem de um elemento é movida
- **onsubmit**: disparado antes de submeter um formulário. Útil para realizar validações. Os dados só são enviados se o evento receber um valor verdadeiro (*true*), valor este que pode ser conseguido como resultado da chamada de uma função que valida as informações do formulário.
- **onunload**: quando a página for fechada

Desta forma, o código da função deve ser declarada dentro da tag <head> ou dentro de um arquivo .js. Os manipuladores de eventos podem ser adicionados a tags ou a elementos (identificado por IDs).

Exemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Título do Navegador</title>
        <meta charset="utf-8" />
        <link rel="stylesheet" type="text/css" href="CSS.css" />
        <script>
            // Declaração da Função:
            function SuaFuncao() {
                /* Comandos */
            }
        </script>
    </head>
    <!-- Função é chamada ao carregar a página: -->
    <body onload="SuaFuncao()">
        <h1 id="titulo">Título na Página</h1>
        <script>
            // Função é chamada ao clicar duas vezes com o mouse sobre o elemento:
            titulo.ondblclick = SuaFuncao;
            // Função é chamada sem evento:
            SuaFuncao();
        </script>
    </body>
</html>
```

- **Variáveis:**

  ✓ Em JavaScript as variáveis podem ser criadas e inicializadas sem declarações formais.
  ✓ Declaração de variáveis:
  `var NOME_VARIAVEL;` (a palavra reservada `var` é opcional)
  ✓ Existem dois tipos de abrangência para as variáveis:
  Global - Declaradas/criadas fora de uma função. As variáveis globais podem ser acessadas em qualquer parte do programa.
  Local - Declaradas/criadas dentro de uma função. Só podem ser utilizadas dentro da função onde foram criadas e precisa ser definida com a instrução var.
  ✓ Com relação à nomenclatura, as variáveis devem começar por uma letra ou pelo caractere sublinhado "_", o restante da definição do nome pode conter qualquer letra ou número. Evite usar caracteres especiais.
  ✓ Tipagem dinâmica de dados: não há necessidade de definir inicialmente o tipo de dados (caractere, lógico, inteiro ou real). Se for necessário, é possível utilizar métodos como **`parseInt()`** (números inteiros) e **`parseFloat()`** (números reais) para converter os dados no momento do processamento.
  ✓ Antes da inicialização, as variáveis possuem o valor especial **undefined**.

- **Operadores:**

  ✓ Operadores de Atribuição:

Operador	Significado
=	Atribuição Simples
+=	Incremental
-=	Decremental
*=	Multiplicativa
/=	Divisória
%=	Modular
++	Incremento
--	Decremento

Os operadores "++" e "--" podem ser utilizados de duas formas diferentes, antes ou depois de uma variável numérica.

  ✓ Operadores Relacionais:

Operador	Significado
==	Igualdade
!=	Diferença
<	Menor
<=	Menor ou Igual
>	Maior
>=	Maior ou Igual

O operador condicional (ternário) "?" é frequentemente usado como um atalho para a instrução if. Sintaxe: `<condição> ? <expressão1> : <expressão2>`

Se a `<condição>` for verdadeira, o operador retorna o valor de `<expressão1>`; se não, ele retorna o valor de `<expressão2>`.

  ✓ Operadores Lógicos:

Operador	Significado
&&	"E"
||	"OU"
!	"NÃO" (inverte valores booleanos)

✓ Operadores Aritméticos:

Operador	Operação
+	Adição e Concatenação de Strings
-	Subtração
*	Multiplicação
/	Divisão
%	Resto de Divisão (Módulo da Divisão)

Para realizar exponenciação é necessário usar o método *pow* do objeto *Math*. Sintaxe:

```
Math.pow(parseInt(VARIAVEL),EXPOENTE);
```

Para obter a raiz quadrada de um número, utiliza-se o método *sqrt* do objeto *Math*. Sintaxe:

```
Math.sqrt(VARIAVEL);
```

Desta forma, o objeto *Math* permite executar tarefas matemáticas com números.

Algumas constantes matemáticas que podem ser obtidas pelo objeto *Math*:
Math.E – Retorna o número de Euler
Math.PI – Retorna o número PI
Math.SQRT2 – Retorna a raiz quadrada de 2

Alguns métodos do objeto *Math*:
Math.round(VALOR) – Retorna o valor arredondado para o seu inteiro mais próximo
Math.ceil(VALOR) – Retorna o valor arredondado para cima
Math.floor(VALOR) – Retorna o valor arredondado para baixo
Math.abs(VALOR) – Retorna o valor absoluto
Math.random() – Retorna um valor randômico entre 0 e 1.

- **Estrutura de Decisão:**

✓ Executa uma parte do código se a condição especificada for verdadeira. Também pode ser especificado um código alternativo para ser executado se a condição for falsa.

```
if ( <condição> ) {
       <bloco de instruções para condição verdadeira>;
       }
else {
       <bloco de instruções para condição falsa>;
       }
```

- **Estrutura de Repetição com variável de controle do tipo contador (FOR):**

✓ Repete uma parte do código um determinado número de vezes utilizando um contador.

```
var i;
for ( <valor_inicial> ; <condição> ; <incremento> ) {
       <bloco de instruções>;
       }
```

Ou

```
for ( var <valor_inicial> ; <condição> ; <incremento> ) {
       <bloco de instruções>;
       }
```

Parâmetro	Significado
valor_inicial	Valor da variável no início loop. Ex.: i = 50
condição	Condição que é verificada para continuar o loop. Ex.: i < 80
incremento	Valor que é adicionado ou subtraído à variável em cada interação. Ex.: i++

- **Estrutura de Repetição (WHILE):**

  ✓ Outro tipo de loop, baseado numa condição ao invés do número de repetições.

```
while ( <condição> ) {
        <bloco de instruções>;
        }
```

- **Comandos**

  ✓ Além das estruturas de controle, o JavaScript apresenta alguns comandos:

  with

  Quando é necessário manipular várias propriedades de um mesmo objeto, a instrução "with" permite fazer isso eliminando a necessidade de digitar o nome do objeto todas as vezes. Sintaxe:

```
with ( <objeto> ) {
        <bloco de instruções>;
        }
```

  Exemplo: Executar uma sequência de operações matemáticas:

```
with (Math) {
    a = PI;          // Número PI
    b = abs(x);      // Valor absoluto da variável x
    c = E;           // Número de Euler
}
```

  break

  Pode ser executado somente dentro de loops "for" ou "while" e tem por objetivo o cancelamento da execução do loop sem que haja verificação na condição de saída do loop, passando a execução a linha imediatamente posterior ao término do loop.

  continue

  Pode ser executado somente dentro de loops "for" ou "while" e tem por objetivo o cancelamento da execução do bloco de comandos passando para o início do loop.

- **Funções:**

  ✓ As funções podem ser definidas como um conjunto de instruções, agrupadas para executar uma determinada tarefa.
  ✓ Para as funções podem ser passadas informações, as quais são chamadas de parâmetros.
  ✓ As funções podem ou não retornar alguma informação, utilizando o comando **return**.
  ✓ Sintaxe de declaração de uma função:

```
function <nome_da_função> ( <parâmetros> ) {
    <bloco de instruções>;
    return ( <valor_de_retorno> );
}
```

- ✓ A chamada de funções é feita da seguinte forma:
  `NomeDaFuncao(<parâmetros>);`

- ✓ Funções são melhor declaradas entre as tags <head> do HTML. Funções são frequentemente chamadas por eventos acionados pelo usuário. Desta forma, as funções são carregadas antes de serem executadas.

• Objetos, Propriedades e Métodos:

- ✓ A linguagem JavaScript é baseada em um paradigma orientado a objetos.
- ✓ Cada elemento de uma página é visto como um objeto. A maneira mais comum de acessar um elemento HTML é usar o **id** do elemento.
- ✓ Os objetos podem ter propriedades, métodos e responder a certos eventos.
- ✓ Um objeto apresenta uma coleção de propriedades, sendo que uma propriedade é uma associação entre um nome (ou chave) e um valor.
- ✓ As propriedades de um objeto definem as características do objeto. Elas são acessadas com uma simples notação de ponto:
  `Objeto.NomeDaPropriedade`
  Exemplo:
  Para obter e alterar o conteúdo de um elemento é possível utilizar a propriedade `innerHTML`.

```
<body>
    <p id="parag">Texto Original</p>
    <script>
        parag.innerHTML = "Novo texto para o parágrafo";
    </script>
</body>
```

- ✓ Um método é uma função associada a um objeto.
- ✓ Além dos objetos que são pré-definidos é possível criar outros objetos.
- ✓ Resumindo, pode-se dizer que os métodos são ações que podem ser executadas em objetos HTML e as propriedades são valores que podem ser definidos, obtidos ou alterados.

- ✓ Alguns objetos pré-definidos:

window
Objeto que representa uma janela aberta no browser que contém certos elementos, como a barra de status.
Alguns métodos do objeto **window**:

alert
Método do objeto **window**. Mostra uma caixa de alerta, seguido de um sinal sonoro e o botão de OK. Sintaxe:
`alert('Mensagem');`

confirm
Método do objeto **window**. Mostra uma caixa de diálogo, seguida de um sinal sonoro e os botões OK e Cancel. Retorna *true* se o usuário escolher OK e *false* se escolher Cancel. Sintaxe:
`<variavel> = confirm('Mensagem');`

prompt

Método do objeto **window**. Mostra uma caixa de diálogo com um campo de texto e os botões OK e Cancel. Sintaxe:

```
<variavel> = prompt('Mensagem', 'Texto default - opcional');
```

window.open() – Abre uma nova janela
window.close() – Fecha a janela atual
window.moveTo() – Movimenta a janela atual
window.resizeTo() – Redimensiona a janela atual
window.history.forward() – Avança o histórico do navegador
window.history.back() – Retrocede o histórico do navegador

window.history.go(1) – Avança o histórico do navegador em 1 página
window.history.go(-3) – Retrocede o histórico do navegador em 3 páginas

document

Objeto que representa a página HTML que está carregado no momento. O objeto **document** é uma propriedade do objeto **window**. Todos os objetos HTML da página são propriedades do objeto **document**.

Algumas propriedades do objeto **document**:

document.lastModified – Data da última modificação da página
document.title – Título da página no navegador

Alguns métodos do objeto **document**:

document.getElementById(id) – Localiza um elemento da página pelo id
document.getElementsByName('name') – Localiza um elemento da página pelo name
document.getElementsByTagName('nome') – Localiza um elemento da página pelo nome da tag
document.getElementsByClassName('nome') – Localiza um elemento da página pelo nome da classe
document.querySelectorAll('seletor') – Localiza um elemento da página pelo nome do seletor
document.write('texto') – Escreve diretamente na página. Se usar o método document.write() depois que um documento HTML tiver sido totalmente carregado, todo o HTML existente será apagado.

location

Objeto que contém informações sobre a URL da página atual. O objeto **location** é uma propriedade do objeto **window**. Desta forma, o objeto **window.location** pode ser usado para obter o endereço de página atual (URL) e para redirecionar o navegador para uma nova página.

- **Strings:**

✓ Strings são usadas para armazenar e manipular textos.
✓ O tamanho de uma string pode ser obtido através da propriedade **length**:

```
nome_variavel.length
```

✓ As posições de uma string podem ser acessadas pelo índice:

```
nome_variavel[indice]
```

✓ Um conteúdo do tipo *string* deve ser delimitado em JavaScript pelos símbolos de apóstrofos (') ou aspas ("). Recomenda-se o uso de apóstrofos, pois as aspas também

são usadas nas tags da linguagem HTML. Se for necessária a utilização destes caracteres como parte da string, utilizar a barra invertida (\) antes do símbolo.
Exemplo: `alert('Cuidado com o uso de \" e \' em uma string.')`

✓ Alguns métodos para trabalhar com strings:

`VARIAVEL.indexOf('texto')` – Retorna a posição da primeira ocorrência de um texto em uma string

`VARIAVEL.lastIndexOf('texto')` – Retorna a posição da última ocorrência de um texto em uma string

`VARIAVEL.search('texto')` – Igual ao método indexOf. Retorna a posição que corresponde a ocorrência de um texto em uma string

`VARIAVEL.slice(pos_início, pos_fim)` – Extrai uma parte de uma string, baseando-se nas posições de início e fim desejados. Retorna a parte extraída em uma nova string. Se o segundo parâmetro for omitido, o método corta o restante da string. Este método aceita posições negativas. Com valores negativos, a contagem das posições inicia pelo fim da string

`VARIAVEL.substring(pos_início, pos_fim)` – Igual ao método slice. Porém, este método somente aceita posições positivas.

`VARIAVEL.substr(pos_início, qtde)` – Extrai uma parte de uma string, baseando-se na posição de início e na quantidade de caracteres desejados. Retorna a parte extraída em uma nova string. Se o segundo parâmetro for omitido, o método corta o restante da string. Este método aceita posições negativas.

`VARIAVEL.replace('texto_pesquisado', 'novo_texto')` – Substitui a primeira ocorrência de um texto por um novo e retorna em uma nova string. Esta função é *case sensitive*.

`VARIAVEL.toUpperCase()` – Converte uma string em letras maiúsculas e retorna em uma nova string

`VARIAVEL.toLowerCase()` – Converte uma string em letras minúsculas e retorna em uma nova string

`VARIAVEL.concat(string1, string2, ...)` – Une duas ou mais strings em uma nova string

`VARIAVEL.charAt(posição)` – Retorna o caractere que ocupa a posição indicada (índice)

- **Arrays:**

✓ Arrays são usados para armazenar múltiplos valores em uma variável.
✓ Um array pode conter muitos valores que podem ser acessados a partir de um número de índice (posição). O índice de um array inicia na posição zero.

✓ Sintaxe para criar um Array:
```
var nome_array = [item1, item2, ...];
```

✓ Sintaxe para obter o valor de uma posição de um array e armazenar em outra variável:
```
var nome_var = nome_array[posição];
```

✓ Sintaxe para modificar o valor de uma posição de um array:
```
nome_array[posição] = NOVO_VALOR;
```

✓ Arrays também podem ser criados como objetos. Os itens podem conter nomes previamente declarados. Neste caso, utilizam-se chaves ao invés de colchetes para criar o array. Exemplo:
```
var pessoas = {primeiroNome:'José', ultimoNome:'Santos', idade:32};
```

Neste exemplo, **pessoas.primeiroNome** retornaria "José".

- ✓ A propriedade **length** retorna a quantidade de elementos de um array:
  nome_array.length

- **Data e Hora:**

  - ✓ Declaração de uma variável que cria um objeto do tipo **Date()**. Sintaxe:

    ```
    var NomeObjetoDate = new Date;
    ```

  - ✓ A partir de um objeto do tipo **Date()** é possível utilizar métodos para obter a data e a hora do sistema. Sintaxes:

    ```
    NomeObjetoDate.getDate()
    NomeObjetoDate.getDay()
    NomeObjetoDate.getMonth()
    NomeObjetoDate.getFullYear()
    NomeObjetoDate.getHours()
    NomeObjetoDate.getMinutes()
    ```

  - ✓ Objeto **Date()** também pode ser usado para obter a data/hora completa, sem necessidade de declaração de uma variável. Exemplo:

    ```
    <body>
        <p id="parag"></p>
        <script>
            parag.innerHTML = Date();
        </script>
    </body
    ```

- **Estilos (CSS):**

  - ✓ A propriedade **style** pode ser utilizada para alterar as propriedades CSS dos elementos HTML. Sintaxe:

    ```
    seletor.style.propriedade = valor;
    ```

  - ✓ A propriedade de estilo **display**, por exemplo, permite esconder um objeto. A mudança pode ser feita de duas maneiras:
    document.getElementById('id_elemento').style.display = 'none';
    seletor.style.display = 'none';

- **Exemplo de Página com Programação Simples em JavaScript:**

  ✓ Ao carregar a página são abertas janelas que solicitam dois números.
  ✓ São realizados 3 cálculos com os valores digitados: Soma, Subtração e Multiplicação.
  ✓ Os resultados são apresentados utilizando cabeçalho tamanho 2.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cálculos com JavaScript</title>
        <meta charset="utf-8" />
        <style type="text/css">
            h2 {
                  font-family: Verdana;
                  text-align: center;
                  color: Navy;
               }
        </style>
    </head>
    <body>
        <h2>Resultados:</h2>
        <h2 id="resp_soma"></h2>
        <h2 id="resp_sub"></h2>
        <h2 id="resp_mult"></h2>
        <script type="text/javascript">
            var NUMERO1;
            var NUMERO2;

            NUMERO1 = prompt('Entre com o valor 1:', 'Digite aqui');
            NUMERO2 = prompt('Entre com o valor 2:', 'Digite aqui');

            resp_soma.innerHTML = "Soma = " + (parseInt(NUMERO1) + parseInt(NUMERO2));
            resp_sub.innerHTML = "Subtração = " + (parseInt(NUMERO1) - parseInt(NUMERO2));
            resp_mult.innerHTML = "Multiplicação = " + (parseInt(NUMERO1) * parseInt(NUMERO2));
        </script>
    </body>
</html>
```