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. Permite criar um conteúdo que se atualiza dinamicamente.
- ✓ 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.
- ✓ O JavaScript diferencia caracteres maiúsculos e minúsculos.
- ✓ Comentários: `/* ... */` ou `//` (para uma linha inteira)
- ✓ O uso do sinal de ponto e vírgula (;) para indicar o final de uma linha de comando é opcional.
- ✓ 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:
  - 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 (interno):

```
<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>
```

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

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

- ✓ **Importante!** Todo o conteúdo HTML de uma página é carregado na ordem em que ele aparece. Ao usar um código em Javascript para manipular elementos que pertencem à página HTML, o código não irá funcionar se a tag <script> for carregada e executada antes dos elementos HTML da página estarem criados e disponíveis. Uma solução é colocar a tag <script> no final da tag <body> da página (antes do fechamento da tag </body>). Com isso, os scripts são carregados depois que todo o conteúdo HTML esteja criado e disponível. Outra solução, que surgiu com a versão HTML5 e funciona apenas para scripts externos, é utilizar o atributo "**defer**" na abertura da tag <script>. Com este atributo, o arquivo do script é carregado paralelamente à página, porém o script é executado apenas após a conclusão do carregamento da página HTML.
- ✓ Em sites maiores, com muitos scripts, carregar os scripts por último pode causar problemas de performance, deixando o site lento. Nestas situações, é possível usar na tag <script> os atributos **async** (que permite baixar o script sem bloquear a renderização da página) ou **defer** (que permite que scripts e HTML sejam carregados de forma simultânea).
- ✓ Quando uma página da web é carregada, o navegador cria um **DOM** (Document Object Model - Modelo de Objetos do Documento) da página. Com o DOM, o JavaScript pode acessar e alterar todos os elementos de um documento HTML. Portanto, o JavaScript obtém todo o poder que precisa para criar um HTML dinâmico. O JavaScript pode:
  - Alterar os elementos HTML na página
  - Alterar os atributos HTML na página
  - Mudar os estilos CSS na página
  - Adicionar novos elementos e atributos HTML na página
  - Remover elementos e atributos HTML existentes
  - Criar eventos HTML na página
  - Reagir a eventos HTML existentes na página
- ✓ Para manipular elementos da página, é comum guardar uma referência ao elemento dentro de uma variável ou constante, usando as funções:

```
document.querySelector('seletor')

document.querySelectorAll('seletor')
```

  O método **querySelector()** retorna o primeiro elemento dentro do documento que corresponde ao(s) seletor(es) CSS especificado(s) como parâmetro. Para retornar todas as correspondências no documento, deve ser usado o método **querySelectorAll()**.

  O parâmetro "seletor" deve ser preenchido da mesma forma que é utilizado no CSS. Pode ser usado tags, classes, identificadores, pseudo-elementos, atributos etc.
- ✓ 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**.

• Funções:

- ✓ Uma função é um conjunto de instruções que executa uma tarefa ou calcula um valor. Para usar uma função, é necessário defini-la (declará-la) em algum lugar dentro do escopo no qual ela será chamada.
- ✓ A declaração de função consiste no uso da palavra-chave **function** seguida por:

  Nome da Função
  Lista de argumentos (parâmetros) para a função, entre parênteses e separados por vírgulas
  Declarações JavaScript que definem a função entre chaves { }
- ✓ A declaração **return** especifica o valor retornado pela função.
- ✓ Parâmetros primitivos (como números e textos) são passados para as funções por valor. Se a função alterar o valor do parâmetro, esta mudança não será refletida globalmente.
- ✓ Se um objeto for passado como parâmetro em uma função (ou seja, um valor não primitivo, tal como Array ou um objeto definido pelo programador) e a função alterar as propriedades do objeto, essa mudança é visível fora da função.

- ✓ Funções também podem ser criadas por uma expressão. Tal função pode ser anônima, ou seja, ela não precisa ter um nome.
- ✓ Uma **Arrow Function** tem uma sintaxe reduzida em comparação ao uso de expressão. **Arrow Functions** são sempre anônimas.
- ✓ As variáveis definidas no interior de uma função não podem ser acessadas de nenhum lugar fora da função, porque a variável está definida apenas no escopo da função. No entanto, uma função pode acessar variáveis e funções definidas no mesmo escopo onde a própria função está definida.
- ✓ Um **método** é uma função invocada por um objeto.
- ✓ A definição de uma função não a executa. Definir a função é simplesmente nomear a função e especificar o que fazer quando a função é chamada. Chamar a função executa realmente as ações especificadas com os parâmetros indicados.
- ✓ Portanto, ao criar 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.

- **Eventos:**

- ✓ Os eventos servem para notificar as mudanças que ocorrem na página que podem afetar a execução do código. Os eventos podem surgir a partir de interações do usuário, como clicar, movimentar o mouse, redimensionar uma janela, entre outros.
- ✓ Diversos eventos que podem ser utilizados para disparar funções. Alguns exemplos:

  `blur`: quando um elemento perder o foco
  `change`: quando um input, select ou textarea tiver o seu valor alterado
  `click`: ao clicar com o mouse
  `dblclick`: ao clicar duas vezes com o mouse
  `focus`: quando um elemento ganhar o foco
  `keypress`: ao pressionar e soltar uma tecla
  `load`: quando o elemento é carregado (página, imagem, script, CSS, entre outros)
  `mousemove`: ao mexer o mouse
  `mouseover`: quando o ponteiro do mouse passa a estar sobre um elemento
  `scroll`: quando a barra de rolagem de um elemento é movida
  `submit`: 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.
  `unload`: quando a página for fechada

- ✓ Em páginas web os eventos são disparados a partir de algum item específico. Os manipuladores de eventos podem ser adicionados a um único elemento, um conjunto de elementos, na página HTML carregada etc.
- ✓ Um manipulador de eventos (event handler) é um bloco de código (geralmente uma função JavaScript definida pelo usuário) que será executado quando o evento for disparado. Manipuladores de eventos são, em alguns casos, chamados de "ouvintes de eventos" (event listeners). Os "ouvintes" escutam o evento acontecendo e o manipulador é o código que roda em resposta a este evento.
- ✓ Há diferentes formas de adicionar manipuladores de eventos para que eles sejam executados quando o evento for disparado.

  Pode ser usada uma propriedade do elemento (exemplo: **`obj_btn.onclick = funAtualiza`**).

  Podem ser usados atributos HTML manipuladores de eventos, também conhecidos como manipuladores de eventos inline. Porém, atualmente, esta é considerada uma prática ultrapassada (exemplo: **`<input type="button" onclick="alert('Mensagem');" />`**

A forma recomendada atualmente é utilizar a função **addEventListener()**. Dentro da função são especificados dois parâmetros: o nome do evento e a chamada da função que se deseja executar: **addEventListener('evento', 'função')**

É possível colocar todo o código dentro da função **addEventListener()**, em uma função anônima. Função anônima é uma função definida sem um nome. Sendo anônima, ninguém conseguirá invocá-la. Mas, como nesta situação a função anônima está associada a um evento, toda vez que o evento ocorrer, ela será chamada.

Para passar valores de parâmetro na chamada da função é necessário usar uma função anônima que chama a função especificada com os parâmetros.

Existe também uma função de contraparte, chamada **removeEventListener()**, que remove um listener adicionado anteriormente.

Também é possível registrar vários manipuladores para um mesmo elemento "ouvinte". Nesta situação, todas as funções associadas serão executadas quando o evento for disparado. Isso seria impossível com as outras opções de manipuladores de eventos, porque qualquer tentativa subsequente sobrescreveria a anterior.

- ✓ Às vezes, dentro de uma função de um manipulador de eventos, é possível encontrar um parâmetro especificado com nome **event**, **evt**, ou simplesmente **e**. Isso é chamado de **event object** e é passado automaticamente para os manipuladores de eventos para fornecer recursos e informações extras.
- ✓ Quando se deseja interromper um evento padrão que seria executado por um elemento (default) pode-se usar a função **preventDefault()** no objeto do evento. O exemplo mais comum é um formulário. Quando se pressiona o botão para submeter, o comportamento natural é que os dados sejam enviados para processamento. Nessa situação, o navegador pode ser redirecionado para uma nova página ou a mesma página pode ser recarregada. A função **preventDefault()** evita este comportamento quando se pressiona o botão.

Exemplo (Funções e Eventos):

Index.html

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Título do Navegador</title>
        <meta charset="utf-8" />
        <link rel="stylesheet" type="text/css" href="CSS.css" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    </head>
    <body>
        <h1>Clique aqui!</h1>

        <!-- O script deve vir por último. Antes de fechar a tag <body> -->
        <script src="MeuScript.js"></script>
    </body>
</html>
```

MeuScript.js

```javascript
const obj_h1 = document.querySelector('h1');

obj_h1.addEventListener('click', funCriarH2);

function funCriarH2() {
  let obj_h2 = document.createElement('h2');
  obj_h2.innerText = 'H2 criado!';
  document.body.appendChild(obj_h2);
}
```

Importante respeitar as letras maiúsculas e minúsculas!

- **Variáveis:**

✓ Em JavaScript as variáveis podem ser declaradas de duas formas: usando let ou var, sendo que let é a forma recomendada.

let - permite declarar variáveis que existem apenas no escopo no qual será utilizada.
var - cria variáveis globais e que podem ser acessadas em qualquer local no arquivo ou na função onde foram declaradas.

✓ As variáveis também podem ser criadas e inicializadas sem declarações formais.

✓ Declaração de variáveis:

```
let NOME_VARIAVEL;
```

✓ 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.

✓ 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. Não pode haver espaços no nome da variável.

✓ 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**.

- **Constantes:**

✓ É uma variável cujo valor é fixo. Uma constante não pode ser redeclarada.

✓ Constantes possuem escopo semelhante às variáveis declaradas usando a palavra-chave let.

✓ Declaração de constantes:

```
const NOME_CONSTANTE inicializador;
```

✓ Toda constante requer um inicializador, ou seja, é preciso especificar um valor para a constante no momento em que ela é declarada. Esse valor não pode ser alterado.

- **Operadores:**

✓ Operadores de Atribuição:

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

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

✓ Operadores Relacionais:

Operador	Significado
==	Valor igual
===	Valor e Tipo iguais
!=	Valor Diferente
!==	Valor ou Tipo diferentes
<	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
**	Exponenciação
/	Divisão
%	Resto de Divisão (Módulo da Divisão)

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

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

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

```
Math.sqrt(VALOR);
```

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

Método para converter um valor usando um número específico de casas decimais:

```
Exemplo com 2 casas decimais: VALOR = VALOR.toFixed(2);
```

Importante respeitar as letras maiúsculas e minúsculas!

- **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.

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

Ou

```
for ( let <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 para Objetos Iterativos (FOR...OF):**

✓ Percorre objetos iterativos, como um array, com instruções para serem executadas a cada objeto distinto. A cada iteração, um valor diferente é atribuído à uma variável.

```
for ( <variável> of <objeto_interativo>) {
        <bloco de instruções>;
}
```

- **Estrutura de Repetição (WHILE):**

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

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

- **Alguns Comandos:**

✓ Além das estruturas de controle, o JavaScript apresenta alguns outros 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.

try...catch

A instrução "try" permite testar erros em um bloco de código. A instrução "catch" permite tratar o erro, ou seja, permite definir um bloco de código a ser executado, se ocorrer um erro no bloco "try". Se um erro ocorrer, ao invés do JavaScript interromper a execução e gerar uma mensagem de erro, ele criará um objeto de Erro com duas propriedades: "name" e "message".

- **Objetos, Propriedades e Métodos:**

✓ A linguagem JavaScript é baseada em um paradigma orientado a objetos.

✓ Quando uma página da web é carregada, o navegador cria um **DOM** (Document Object Model - Modelo de Objetos do Documento) da página.

✓ 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.

✓ A maioria dos atributos das tags HTML tornam-se automaticamente propriedades de objetos DOM.

✓ As propriedades de um objeto definem as características do objeto. Elas podem ser acessadas de algumas formas:

```
Objeto.NomeDaPropriedade
Objeto["NomeDaPropriedade"]
Objeto[expressão]        * A expressão deve resultar no nome da propriedade
```

Exemplo:

Para alterar o conteúdo de um elemento HTML é possível utilizar a propriedade **innerText.**

```
<body>
    <p id="parag">Texto Original</p>
    <script>
        const obj_parag = document.querySelector('#parag');
        obj_parag.innerText = "Novo texto para o parágrafo";
    </script>
</body>
```

Outra possibilidade de codificação, usando um link e trocando outras propriedades:

```
<body>
    <a id="meu_link" href="Index.html" target="_self">Link Original</p>
    <script>
        document.getElementById("meu_link").innerHTML = "Novo Link";
        document.getElementById("meu_link").href = "http://google.com";
        document.getElementById("meu_link").target = "_blank";
    </script>
</body>
```

- ✓ Os métodos getAttribute(), getAttribute() e removeAttribute() também podem ser usados para obter e manipular atributos de tags HTML. Exemplo: **Objeto.getAttribute('alt')**

- ✓ Algumas propriedades:

Objeto.checked

Define ou retorna o estado de alguns tipos de elementos que podem ser selecionados. Esta propriedade reflete o atributo HTML "checked". Valores: true (o elemento está selecionado) ou false (o elemento não está selecionado).

Objeto.classList

Retorna os nomes das classes de um elemento. Esta propriedade é útil para adicionar, remover e alternar classes CSS em um elemento. É possível modificá-la usando os métodos add() e remove().

Objeto.innerHTML

Define ou retorna o conteúdo HTML de um elemento. Retorna outras tags HTML que eventualmente estejam no elemento, como por exemplo, uma tag dentro de um <p>.

Objeto.innerText

Define ou retorna somente o conteúdo de texto do elemento especificado e o conteúdo de texto de todos os seus descendentes. Se a propriedade innerText for definida, quaisquer tags contidas no objeto serão removidas e substituídas por um único elemento contendo a string especificada.

Objeto.selected

Define ou retorna o estado selecionado de uma opção. Valores: true (a opção está selecionada) ou false (a opção não está selecionada).

Objeto.value

Para elementos do tipo <button>, <input> e <option>, o atributo "value" especifica o valor inicial do elemento. Portanto, para estes tipos de elementos, esta propriedade define ou retorna o conteúdo do atributo "value" de um elemento HTML. Em listas ordenadas, a propriedade "value" pode ser usada em um elemento <li> para definir o número a partir do qual o identificador da lista ordenada irá iniciar (quando não se deseja que a lista ordenada inicie em 1). Para elementos do tipo <progress> e <meter>, o atributo value especifica o avanço da barra medidora.

Importante respeitar as letras maiúsculas e minúsculas!

- ✓ Diferença entre **propriedades** e **atributos**: os atributos são definidos nas tags HTML. As propriedades são acessadas a partir do DOM (Document Object Model). É a partir do DOM que o JavaScript pode acessar e alterar todos os elementos de um documento HTML. Desta forma, pode-se dizer que as propriedades são associadas a objetos JavaScript enquanto os atributos são associados a tags HTML.
- ✓ 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

Importante respeitar as letras maiúsculas e minúsculas!

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

Importante respeitar as letras maiúsculas e minúsculas!

Alguns métodos do objeto **document**:

document.createElement('nome_tag') – Cria o elemento HTML (tag) especificado. O método retorna um objeto que representa o elemento criado. Depois que o elemento for criado, os métodos **Objeto.appendChild()** ou **Objeto.insertBefore()** devem ser usados para inserir o elemento no documento.

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.querySelector('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.

Importante respeitar as letras maiúsculas e minúsculas!

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.

- **Alguns Métodos:**

`Objeto.appendChild('nome_elemento')` – Adiciona um elemento como último filho do elemento pai. Também é possível usar este método para mover um elemento de um elemento para outro.

`Objeto.removeChild('nome_elemento')` – Remove um elemento filho especificado do elemento pai.

`Objeto.replaceChild('elemento_novo','elemento_removido')` – Substitui um elemento filho por um novo.

`Objeto.insertBefore('elemento_novo','elemento_existente')` – Insere um novo elemento filho antes de um elemento filho existente especificado.

`Objeto.getAttribute('nome_atributo')` – Retorna o valor de um atributo especificado no elemento. Se o atributo não existir, o valor retornado será nulo ou "" (string vazia).

`Objeto.setAttribute('nome_atributo', 'valor')` – Adiciona um novo atributo ou modifica o valor de um atributo existente num elemento específico.

`Objeto.removeAttribute('nome_atributo')` – Remove um atributo existente num elemento específico.

Importante respeitar as letras maiúsculas e minúsculas!

- **Strings:**

✓ Strings são usadas para armazenar e manipular textos.

✓ O tamanho de uma string pode ser obtido através da propriedade **length**:

```
var_string.length
```

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

```
var_string[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:

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

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

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

`var_string.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

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

`var_string.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.

`var_string.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*.

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

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

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

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

Importante respeitar as letras maiúsculas e minúsculas!

- **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:

```
let nome_array = [item1, item2, ...];
```

✓ Sintaxe para obter o valor de uma posição de um array e armazenar em outra variável:

```
let 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:

```
let 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()
```

Importante respeitar as letras maiúsculas e minúsculas!

✓ 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>
       const obj_parag = document.querySelector('#parag');
       obj_parag.innerText = Date();
    </script>
</body
```

- **Estilos (CSS):**

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

✓ Não é possível definir estilos atribuindo uma string à propriedade de estilo. Por exemplo, `Objeto.style = "color: red;"`. Para definir o estilo de um elemento, deve ser adicionada a propriedade CSS, após a propriedade "style", seguido pelo respectivo valor. Exemplo: `Objeto.style.color = 'red'`.

✓ Importante: A sintaxe JavaScript para definir propriedades CSS é ligeiramente diferente da sintaxe CSS. Exemplo: Em JavaScript utiliza-se **backgroundColor** em vez **background-color**.

Sintaxe:

```
Objeto.style.propriedade = valor;
```

✓ A propriedade de estilo **display**, por exemplo, permite esconder um objeto. A mudança pode ser feita de diferentes maneiras. Algumas possibilidades:

```
document.querySelector('.classe').style.display = 'none';
document.getElementById('id_elemento').style.display = 'none';
Objeto.style.display = 'none';
```

Importante respeitar as letras maiúsculas e minúsculas!

✓ É recomendado usar a propriedade "style" em vez do método `Objeto.setAttribute("style", "...")`, porque a propriedade "style" não sobrescreverá outras propriedades CSS que podem ser especificadas no atributo "style".

- **Palavra-chave "this":**

✓ Esta palavra-chave se refere ao objeto ao qual pertence. Seu valor é estabelecido segundo o contexto de execução no qual está inserido. Pode-se dizer que "this" é o sujeito do código em execução. O "this" sempre se refere a um objeto singular e normalmente é usado dentro de uma função ou método.

- ✓ Em uma função JavaScript, o proprietário da função é a ligação padrão para o "this". Ou seja, o "this" faz referência ao "objeto antecedente" (ou objeto invocador).

- **Função eval():**

- ✓ A função eval() avalia ou executa um argumento. O argumento da função eval() é uma string.

```
eval('argumento')
```

- ✓ Se o argumento for uma expressão, a função avalia a expressão. Se o argumento for uma ou mais instruções JavaScript, eval() executa as instruções.
- ✓ Se uma expressão aritmética for construída como uma string, é possível usar eval() para calculá-la posteriormente.

- **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.

Index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cálculos com JavaScript</title>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />

        <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 src="MeuScript.js"></script>
    </body>
</html>
```

MeuScript.js

```
const obj_resp_soma = document.querySelector('#resp_soma');
const obj_resp_sub = document.querySelector('#resp_sub');
const obj_resp_mult = document.querySelector('#resp_mult');

let NUMERO1;
let NUMERO2;

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

obj_resp_soma.innerText = "Soma = " + (parseInt(NUMERO1) + parseInt(NUMERO2));
obj_resp_sub.innerText = "Subtração = " + (parseInt(NUMERO1) - parseInt(NUMERO2));
obj_resp_mult.innerText = "Multiplicação = " + (parseInt(NUMERO1) * parseInt(NUMERO2));
```

Importante respeitar as letras maiúsculas e minúsculas!