

CEUB

EDUCAÇÃO SUPERIOR

Desenvolvimento de Interfaces - DI

ceub.br

Aula 07 – JavaScript

JavaScript

Introdução

Como visto anteriormente, as linguagens HTML e CSS são fundamentais para a criação de páginas web.

O foco do HTML é o conteúdo enquanto o foco da CSS é a formatação das páginas.

As linguagens HTML e CSS não são linguagens de programação, sendo que para resolver determinados problemas são necessárias linguagens de programação.

Introdução

JavaScript é uma das três linguagens Web que todo desenvolvedor deveria aprender:

- 1. HTML para definir o *conteúdo* de páginas Web;**
- 2. CSS para especificar o *layout* e a formatação das páginas Web;**
- 3. JavaScript para programar o *comportamento* das páginas Web.**

Introdução

- JavaScript é uma linguagem de programação que roda no lado cliente (no navegador do usuário).
- Porém atualmente ela também pode rodar no lado do servidor, como exemplo pode ser citado o **Node.js** que rodam JS no servidor para diversos propósitos.
- Usando a linguagem JavaScript é possível construir páginas mais dinâmicas e interativas.
- O foco do JS é **implementar o comportamento** ou a inteligência das páginas Web.

Introdução

- **JavaScript é uma linguagem de programação interpretada client-side com tipagem dinâmica (linguagem de script);**
- **Linguagem de programação – permite a criação de rotinas (conjuntos de instruções) com finalidades específicas. Ex: validar entradas em um formulário, alterar textos, tags ou propriedades CSS de uma página etc.**
- **Interpretada – não é compilada, isto é, o código escrito pelo desenvolvedor é lido e executado pelo interpretador (neste caso, o navegador).**

Introdução

- O JavaScript pode ser colocado em qualquer lugar dentro de uma página web.
- O mais comum é encontrarmos o código dentro da tag `<head>`.
- Para colocar código JavaScript, utilizamos a tag `<script>`

JavaScript

Hoje o maior mantenedor da linguagem JavaScript é a Fundação Mozilla;



Encontramos ótimos materiais e tutoriais sobre JavaScript na W3School, mas também encontramos referência completa do JavaScript no site do Mozilla:

**[https://developer.mozilla.org/
en/docs/JavaScript](https://developer.mozilla.org/en/docs/JavaScript)**

JavaScript

Os principais padrões a destacar são:

– A Linguagem Núcleo:

- **ECMAScript** (Versão 14, de Junho de 2023);
- Padrão mantido por ECMA International-Associação Industrial de padronização de tecnologias da Informação e Comunicação;

– DOM:

- ***Document Object Model;***
- Define a Interface da Linguagem com o Browser;
- Padrão mantido por **W3C**;

JavaScript

De acordo com a documentação do HTML a tag `<script>` serve para “definir um script client-side (JavaScript)”.

Elá pode conter um código JS ou apontar para um arquivo externo usando o atributo `src`. Exemplos:

Vinculação Interna

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Exemplo Vinculação Interna</title>
    <script type="text/javascript">
        //Código Javascript --Vinculação Interna
        //O mais recomendado é que os arquivos sejam sempre acrescentados antes do fechamento da tag body
    </script>
</head>
```

Vinculação Externa

```
<body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/2.9.2/umd/popper.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/5.1.0/js/bootstrap.min.js"></script>
</body>
```

Código JavaScript

- **JavaScript é *case sensitive*, ou seja, isso significa que duas variáveis com nomes diferentes, mas com as mesmas letras, mas em maiúsculas e minúsculas diferentes, serão consideradas variáveis diferentes.**
- **Declarações em JavaScript podem ou não terminar com o ponto-e-vírgula;**
- **Comentários de linha são do tipo
`// comentário`**
- **Comentários de bloco são do tipo
`/* comentário */`**

JavaScript - Variáveis

- **Variáveis podem ser declaradas com as palavras var, const ou let;**
- **var** - Quando definidas com **var**, fora de funções, possuem escopo global e podem ser acessadas dentro e fora da função; Quando definidas dentro de funções, possuem escopo local a toda a função;
- **let** - Declara uma variável local de escopo do bloco, opcionalmente, inicializando-a com um valor. Ou seja, só podem ser acessadas apenas dentro do bloco de código em que foi definida, como um for, um if-else, etc;
- **const** - Declara uma constante de escopo de bloco, apenas de leitura.

JavaScript - Variáveis

```
<!DOCTYPE html>
<html class="no-js">
<head>
</head>
<body>
    <script>
        var a = 8;
        var b = 2;
        var h = 3; // Declara três variáveis inteiros
        var area = ((b * h) / 2);
        document.writeln('A área de um triângulo cuja base é 2 e a altura é 3 é = ' + area + '<br>');
        document.writeln();
        var str = 'Programação com JavaScript'; // str é uma string var A = true; // A é uma variável booleana
        if (b < h) {
            let d = a + b; // d poderá ser acessada apenas dentro deste bloco
            document.writeln('Valor da variável d é = ' + d);
        }
    </script>

    <script src="" async defer></script>
</body>
</html>
```

JavaScript - Saída de Dados

JavaScript pode gerar e mostrar dados de diferentes maneiras:

Escrevendo no conteúdo de um elemento HTML por meio da propriedade innerHTML (há propriedades similares como innerText e textContent – pesquisar)

Escrevendo no documento HTML propriamente dito usando document.write('conteudo');

Escrevendo em uma caixa de mensagens usando window.alert('mensagem');

Escrevendo no console do navegador, para fins de desenvolvimento/debug, usando console.log('info').

JavaScript –Exemplo de Saída de Dados

```
<!DOCTYPE html>
<html>
<body>
<h1>Linguagem JavaScript</h1>
<p>Utilizando document.write para gerar conteudo</p>
<script>
    console.log('Escrevendo no console do navegador...');
    for (var i = 0; i < 10; i++)
        document.write('<h1>Texto gerado por funcao JavaScript </h1>');
</script>
<button type="button" onclick="window.alert('Obrigado!')">Clique aqui!</button>
</body>
</html>
```

- **window, document e console** são objetos;
- **alert, write e log** são métodos dos respectivos objetos;
- Para exibição do console do navegador, tecle **<F12>** e procure pela aba *console*.

Caixas de diálogos - JavaScript

JavaScript disponibiliza 3 tipos de caixas de diálogo:

- **alert()**
- **prompt()**
- **confirm()**

Caixas de diálogos - JavaScript

confirm()

- Esta caixa de diálogo possui dois botões. Um de OK e outro Cancelar. Pode-se usar esta função para solicitar uma decisão do usuário.

```
<script>
```

```
    opcao = confirm("Deseja prosseguir?");

    if(opcao)
        alert("Você clicou em OK");
    else
        alert("Você clicou em CANCELAR");
```

```
</script>
```

Operadores Aritméticos, Relacionais e Lógicos

Operadores Aritméticos

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira (módulo)
++	Incremento
--	Decremento

Operadores Relacionais e Lógicos

Operador	Significado
==	Comparação por igualdade
==	Comparação por igualdade, incluindo valor e tipo
!=	Diferente
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a
&&	“E” lógico
	“Ou” lógico
!	Negação lógica

OBS: O Operador **+** também pode ser utilizado para concatenar *strings*

Operador de Adição e Concatenação

- Deve-se utilizar o operador + com atenção, pois a operação executada (soma ou concatenação) depende do tipo dos operandos;
- JavaScript avalia as expressões da esquerda para a direita;
- Ao avaliar um par de operandos, se um deles for string e o outro numérico, então o numérico será convertido para string.
- Exemplos

```
x = 5 + 5;           // x terá o valor 10
y = "5" + 5;         // y terá a string „55“
z = "Hello" + 5;    // z terá a string „Hello5“
w = 2 + 4 + "5";   // w terá a string „65“
p = "5" + 2 + 4;   // p terá a string „524“
```

Diferença dos Operadores == e ===

O operador **==** compara apenas os valores dos operandos. Se os operandos forem de tipos diferentes, uma conversão é realizada e os valores convertidos são comparados;

O operador **====** compara o valor e o tipo dos operandos. A comparação de operandos de tipos diferentes sempre resulta em *falso*.

Exemplos:

```
1 == "1";      // true
1 == true;     // true
1 === "1";     // false
1 === true;    // false
```

```
var x = 10;
var y = "10";           // true
x == y;
var x = 10;
var y = "10";
x === y;               // false
```

Operadores de Atribuição

Operador	Significado	Exemplo
=	Atribuição	<code>var x = 0;</code> <code>// atribui o valor 0 a x</code>
+=	Atribuição com soma	<code>var x += y;</code> <code>// equivalente a: x = x + y</code>
-=	Atribuição com subtração	<code>var x -= y;</code> <code>// equivalente a: x = x - y</code>
*=	Atribuição com multiplicação	<code>var x *= y;</code> <code>// equivalente a: x = x * y</code>
/=	Atribuição com divisão	<code>var x /= y;</code> <code>// equivalente a: x = x / y</code>
%=	Atribuição com módulo	<code>var x %= y;</code> <code>// equivalente a: x = x % y</code>

Operadores relacionais

Os operadores relacionais são utilizados em expressões condicionais para a comparação do valor de duas expressões:

> → Maior que

>= → Maior ou igual à

< → Menor que

<= → Menor ou igual à

== → Igual à

!= → Diferente de

Estruturas Condicionais e de Repetição

```
if (expressão) {  
    // operações  
}  
  
if (expressão) {  
    // operações caso verdadeiro  
}  
else {  
    // operações caso falso  
}  
  
if (expressao1) {  
    // operações 1  
}  
else if (expressao2) {  
    // operações 2  
}  
else {  
    // operações 4  
}  
  
for (var i = 0; i < 10; i++)  
{  
    // operações  
    // operações  
}  
  
while (expressao)  
{  
    // operações  
    // operações  
}  
  
do {  
    // operações  
    // operações  
} while (expressao)
```

Estrutura Condicional *switch-case*

O comando **switch-case** é útil quando temos uma sequência longa de **if-else's**. Essa estrutura permite comparar uma expressão com diversas condições possíveis:

```
switch (expressão) {  
    case valor1://Instruções executadas quando o resultado da expressão for igual  
    á valor1  
        [break;]  
    case valor2://Instruções executadas quando o resultado da expressão for igual  
    á valor2  
        [break;]  
    ...  
    case valueN://Instruções executadas quando o resultado da expressão for igual  
    á valorN  
        [break;]  
    default:  
        //Instruções executadas quando o valor da expressão é diferente de todos os cases  
        [break;]  
}
```

Operador condicional (ternário)

O operador condicional (ternário) é o único operador JavaScript que possui três operandos. Este operador é frequentemente usado como um atalho para a instrução if.

condition ? expr1 : expr2

Parâmetros

condition

Uma expressão que é avaliada como true ou false.

expr1, expr2

Expressões com valores de qualquer tipo.

Estrutura de Repetição for

O comando **for** representa um mecanismo de repetição, onde é necessário definir 3 condições:

- (1) **início da contagem;**
- (2) **condição de parada;**
- (3) **incremento/decremento**

```
<script>
    for (i=0; i<10; i++){
        document.write("<h2>" + i + "</h2>");
    }
</script>
```

Arrays (Vetores) - JavaScript

Arrays (Vetores)

Em JavaScript, **vetores (arrays)** podem ser definidos colocando-se os elementos entre colchetes (separados por vírgula);

O primeiro elemento do **array** possui índice 0;

Os vetores são tratados como objetos. Por exemplo, o número de elementos pode ser resgatado por meio da propriedade **length**;

Exemplo:

```
var vet = [5,'a',3,'java',2];
console.log(vet[1]); // a saída será o caractere 'a'
console.log(vet[2]); // a saída será o número 3
```

Arrays (Métodos)

Os métodos são funções pré-estabelecidas usadas para realizar operações em objetos, como arrays, strings e objetos.

Em JavaScript, os métodos são chamados usando a sintaxe de ponto. Por exemplo, o método push() é usado para adicionar elementos a um array.

Para usar o método push(), você chamaria o método em um array existente usando a sintaxe de ponto:

Arrays (Principais Métodos)

- **push():** Adiciona um elemento ao final do array.
- **pop():** Remove o último elemento do array.
- **shift():** Remove o primeiro elemento do array.
- **unshift():** Adiciona um elemento ao início do array.
- **slice():** Retorna um novo array com uma parte do array original.
- **concat():** Concatena dois ou mais arrays.

Arrays (Principais Métodos)

- **join():** Junta os elementos do array em uma string.
- **sort():** Ordena os elementos do array.
- **reverse():** Inverte a ordem dos elementos do array.
- **filter():** Retorna um novo array com os elementos que atendem a um determinado critério.
- **map():** Aplica uma função a cada elemento do array e retorna um novo array com os resultados.
- **reduce():** Aplica uma função cumulativa a cada elemento do array e retorna um único valor.

Exemplos

Document Object Model (DOM)

Document Object Model

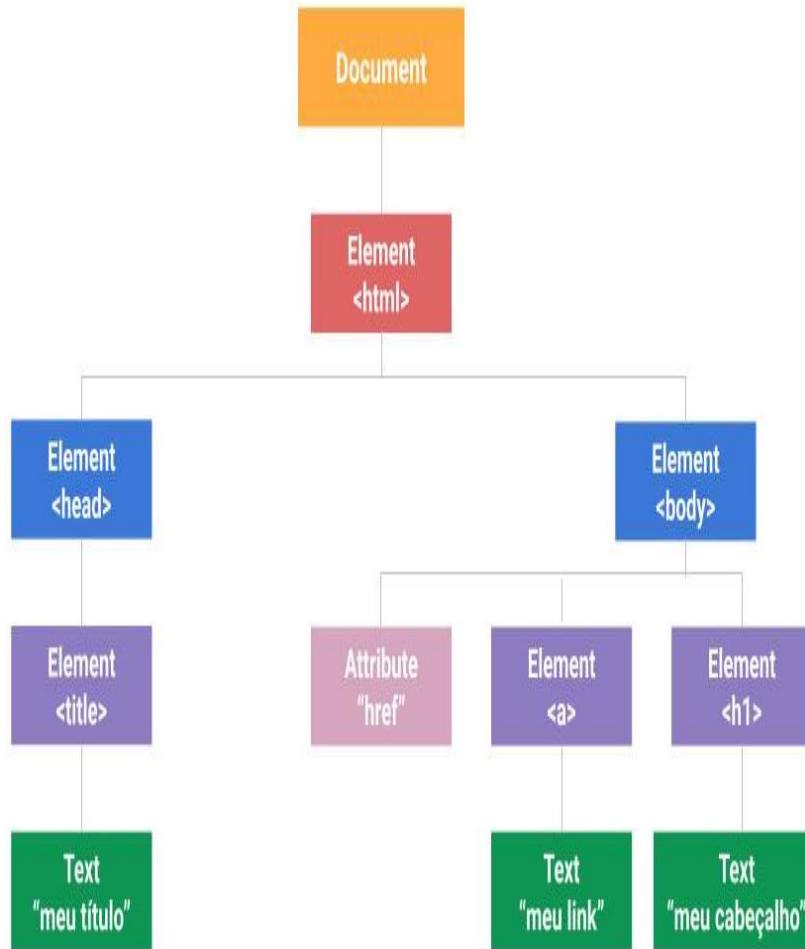
DOM - *Document Object Model*

É um modelo utilizado pelos navegadores de Internet no qual um documento HTML é representado como uma coleção hierárquica de objetos;

Em outras palavras, DOM é uma representação em estrutura de árvore de todos os elementos de uma página Web;

Document Object Model (DOM)

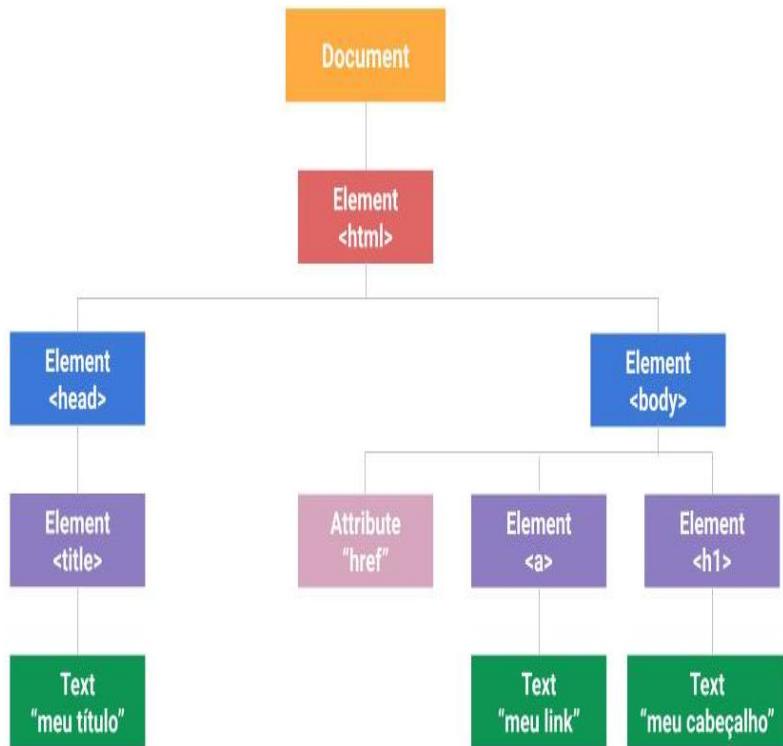
Assim, tudo no documento HTML corresponde a um nó:



- **Documento propriamente dito é um nó (do tipo *document*);**
- **Todos os elementos HTML, como <p>, <a>, <h1>, etc., são nós (do tipo *element*);**
- **Todos os atributos HTML são nós (do tipo *attribute*)**
- **O conteúdo dos elementos HTML são nós (do tipo *text*)**
- **E até mesmo os comentários são nós (do tipo *comment*)**

Document Object Model (DOM)

Hierarquia de nós na estrutura DOM



- **Nó Root:** nó representando o elemento raiz `<html>`
- **Nó Filho:** nó representando um elemento diretamente dentro de outro
- **Nó Pai:** nó representando o elemento que contém o nó filho
- **Nós Irmãos:** nós representando elementos filhos do mesmo pai

Document Object Model (DOM)

Busca na árvore DOM

A estrutura de objetos do documento pode ser acessada por meio do objeto *document*, que é o nó raiz da hierarquia.

`document.querySelector("seletor CSS")`: retorna o primeiro elemento que casa com o seletor CSS especificado;

Document Object Model (DOM)

Algumas formas de resgatar elementos da estrutura DOM:

- **`document.getElementById("id_do_elemento")`**: retorna o elemento HTML por meio do id do elemento;
- **`document.getElementsByName("nome_do_elemento")`**: retorna a coleção de elementos que possuem o atributo name igual a “*nome do elemento*”;
- **`document.getElementsByTagName("tag_name")`**: retorna a coleção de elementos cujo nome da tag é “*tag_name*”;
- **`document.getElementsByClassName("class_name")`**: retorna a coleção de elementos que utilizam a classe CSS “*class_name*”;

Document Object Model (DOM)

Algumas formas de resgatar elementos da estrutura DOM:

- **document.querySelector(“seletor CSS”):** retorna o primeiro elemento que casa com o seletor CSS especificado;

Document Object Model (DOM)

Outros métodos para manipulação da estrutura DOM

- **document.createElement**
- **document.createTextNode**
- **node.childNodes**
- **node.parentNode**
- **node.appendChild**
- **node.removeChild**
- **node.hasChildNodes**
- **node.cloneNode**
- **node.firstChild node.lastChild node.nextSibling
node.previousSibling**

Materiais complementares recomendados:

https://www.w3schools.com/js/js_htmldom.asp

<https://www.impressivewebs.com/10-essential-dom-methods-techniques-for-practical-javascript/>

https://www.w3schools.com/jsref/met_node_appendchild.asp

Exemplos do Uso do DOM

Collections do Objeto document

Alguns elementos do documento HTML (como formulários, links e imagens) também podem ser acessados por meio de propriedades especiais denominadas collections;

- **document.forms:** retorna uma coleção com todos os formulários da página;
- **document.images:** retorna uma coleção com todas as imagens (``) da página;
- **document.anchors:** retorna uma coleção com todos links (`<a>`) da página;

Exemplos:

- Ao invés de indicar uma função JavaScript para tratar um evento diretamente no código HTML, como em:

```
<input type="button" onclick="funcaoJavaScript()">
```

- Pode-se utilizar o método ***addEventListener*** para associar a função tratadora do evento fora do HTML, como em:

```
var btn = document.getElementById("myBtn"); btn.addEventListener("click",  
funcaoJavaScript());
```

- Umas das vantagens de se utilizar a segunda forma é a separação do código JavaScript do HTML.

Validação de Formulários

Validação de Formulários

- **JavaScript pode ser utilizada para validar formulários no lado *cliente*, pelo navegador;**
- **Pode-se utilizar o atributo *onSubmit* do elemento <form> em conjunto com uma função JavaScript para realizar a validação;**
- **Neste caso, a função deve verificar o conteúdo dos campos e retornar true quando todos os campos forem válidos ou false quando algum campo for inválido;**
- **O formulário será submetido apenas quando o valor de retorno da função for verdadeiro;**

Validação de Formulários - Exemplo

```
<!DOCTYPE html>
<html>
<head><title>Validando Formulários com JavaScript</title>
<script>
    function validaForm() {
        var usuario = document.forms["form1"]["usuario"].value;
        if (usuario == null || usuario == "") {
            alert("O campo usuário deve ser preenchido");
            return false;
        }
        var senha = document.forms["form1"]["senha"].value;
        if (senha == null || senha == "") {
            alert("O campo senha deve ser preenchido");
            return false;
        }
        return true;
    }
</script>
</head>
<body>
<form name="form1" action="login.php" onSubmit="return validaForm()" method="post">
    Usuário: <br> <input type="text" name="usuario"> <br>
    Senha: <br> <input type="password" name="senha"> <br><br>
    <input type="submit" value="Enviar">
</form>
</body>
</html>
```

Eventos JavaScript

Eventos JavaScript

Em JavaScript, eventos são ações ou ocorrências que acontecem no navegador, como o clique de um mouse, uma tecla pressionada, o carregamento de uma página, a alteração de um campo de entrada, entre outras interações do usuário ou eventos do navegador.

Os eventos são fundamentais para a criação de aplicações web interativas e dinâmicas.

Eventos JavaScript

Funções para tratar eventos podem ser indicadas, na maioria dos casos, de duas formas:

- Utilizando propriedades de eventos

```
function funcaoSaudacao() {  
    alert("Hello!");  
    console.log("Hello!");  
}  
  
// Exemplo 1 - O evento load ocorre quando a página inteira é carregada  
window.onload = funcaoSaudacao;
```

Eventos JavaScript

Funções para tratar eventos podem ser indicadas, na maioria dos casos, de duas formas:

- Utilizando o método **addEventListener** -

```
function funcaoSaudacao() {  
    alert("Hello Word!");  
    console.log("Hello!");  
}  
// o primeiro parâmetro é o nome do evento e não tem 'on'  
// o segundo parâmetro define a função para tratar o evento,  
// também conhecida como função de callback  
window.addEventListener("load", funcaoSaudacao);  
//2ª Forma  
window.addEventListener("load", function () {  
    alert("Hello!");  
    console.log("Hello!");  
});  
//Ou ainda usando funções anônimas  
window.addEventListener("load", function () {  
    alert("Hello!");  
    console.log("Hello!");  
});
```

Eventos JavaScript

Alguns eventos frequentemente utilizados:

- **Onclick:** O evento onclick é utilizado para associar uma ação que deve ser executada quando um evento de clique ocorrer;
- **onMouseEnter:** Quando o usuário ‘entra’ com o ponteiro do mouse sobre o elemento;
- **onMouseLeave:** Quando o usuário ‘retira’ o ponteiro do mouse do elemento;
- **onMouseDown:** Quando o usuário pressiona um botão do mouse sobre o elemento;
- **onMouseUp:** Quando o usuário solta o botão do mouse sobre o elemento;
- **onMouseOver:** Quando o usuário move o ponteiro do mouse sobre o elemento;
- **onChange:** Em alguns elementos, quando o seu valor (value) muda;
- **onFocus:** Quando o elemento recebe foco.

Orientação a Objetos com JavaScript

Orientação a Objetos com JavaScript

A orientação a objetos é um paradigma de programação que permite aos desenvolvedores criar software pensando em termos de objetos.

Um objeto é uma entidade que possui estado e comportamento.

O estado de um objeto é representado por seus atributos, e o comportamento de um objeto é representado por seus métodos.

Orientação a Objetos com JavaScript

Em JavaScript, a orientação a objetos é implementada por meio de classes.

Uma classe é um molde que define os atributos e métodos de um objeto. Para criar um objeto, você cria uma nova instância de uma classe.

Porém o mais comum é que a orientação a objetos em JavaScript seja feita usando protótipos.

Protótipos são um mecanismo que permite que objetos compartilhem propriedades e métodos.

Definindo Funções

```
function nomeDaFuncao(par1, par2, par3) {  
    // operações  
    // operações  
    // operações  
}  
  
function adicionar(a, b) {  
    return a + b;  
}  
  
const resultado = adicionar(3, 4);  
console.log(resultado); // Imprime 7
```

Quando ‘return’ não é utilizada, o valor `undefined` é automaticamente retornado.

OBS: Funções em JavaScript não precisam necessariamente retornar um valor. Caso a declaração ‘return’ não seja utilizada, o valor `undefined` será automaticamente retornado.

Definindo Funções – Argumentos Opcionais

Caso um argumento deixe de ser fornecido ao chamar a função, o parâmetro correspondente receberá o valor undefined;

Exemplo:

```
function potencia(base, expoente) {  
    if (expoente == undefined)  
        expoente = 2;  
  
    var resultado = 1;  
    for (var i = 0; i < expoente; i++)  
        resultado = resultado * base;  
    return resultado;  
}  
  
console.log(potencia(2, 5)); // a saída será 32  
console.log(potencia(3)); // a saída será 9 (o exp. padrao é 2)
```

Funções e Escopo de Variáveis

- **Variáveis definidas dentro de funções têm escopo local e podem ser acessadas apenas dentro delas;**
- **Variáveis definidas fora das funções têm escopo global e podem ser acessadas por qualquer script ou função da página Web;**
- **Toda variável global definida no código JavaScript pode ser acessada como uma propriedade do objeto window:**
- **window.nomeDaVariavel;**

Funções Definidas em Arquivo Externo - Exercício

1- A partir do arquivo JavaScript abaixo crie um arquivo JavaScript externo com nome script.js. Após cria-lo você deverá vinculá-lo a um evento onClick de um botão em uma página html;

```
/* arquivo script.js */
function fatorial(n) {
    var total = 1;
    for (var i = 1; i <= n; i++) {
        total = total * i;
    }
    return total;
}

function testeFatorial() {
    var num = prompt("Informe um numero inteiro positivo: ");
    var numInt = parseInt(num); // converte a string em inteiro
    var fat = fatorial(numInt);
    document.write('O fatorial É: ' + fat);
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Exemplo Fatorial</title>
    <script src="script.js"></script>
</head>
<body>
    <input type="button" onclick="testeFatorial()" value="Clique para calcular">
</body>
</html>
```

Funções Anônimas

Em JavaScript, funções anônimas são funções que não têm um nome associado.

Elas são definidas diretamente como expressões de função e podem ser usadas em qualquer lugar onde você normalmente usaria uma função nomeada.

Funções anônimas são frequentemente usadas como argumentos de outras funções de ordem superior, como map, filter, reduce, addEventListener, entre outras.

Funções Anônimas

Em JavaScript, funções anônimas são funções que não têm um nome associado.

Elas são definidas diretamente como expressões de função e podem ser usadas em qualquer lugar onde você normalmente usaria uma função nomeada.

Exemplo:

```
//Função Tradicional
function somaDoisNumeros(a, b) {
    return numero + numero;
};
```

```
//Função Anônima
const soma = function (a, b) {
    return a + b;
};
```

Arrow function =>

Uma função arrow é uma forma concisa de criar uma função em JavaScript. Ela é representada por uma seta (=>) – (lambda) seguida de um corpo de função

Elas são amplamente usadas em código JavaScript moderno devido à sua simplicidade e legibilidade.

Em resumo ela define funções sem utilizar a palavra function. Exemplo:

```
//Função Tradicional
function somaDoisNumeros(a, b) {
    return numero + numero;
};
```

```
var soma = (a, b) => a + b;

//Função com uma única declaração dispensa as chaves
window.onload = () => alert('Olá Mundo...');

//Arrow function também pode ter parâmetros
window.onload = (e) => alert('Objeto:' + e.target);

//Arrow function com um único parâmetro não precisa dos parênteses
window.onload = e => alert('Objeto:' + e.target);
```

Arrays - Métodos Funcionais e poderosos)

Map, Reduce e Filter são três métodos poderosos que podem ser usados para manipular arrays em JavaScript.

Esses métodos permitem que você aplique uma função a cada elemento de um array, retornando um novo array com os resultados.

Esse métodos permitem manipular um array de um modo funcional em Javascript

Método Map

O **método map()** tem o objetivo de executar uma função em cada item de um array não o sobrescrevendo-o mas sim criando um novo array após a manipulação, ou seja, não sobrescreve o array original.

O método **map()** pode ser usado como uma alternativa para laços de repetição, que iteram sobre o array.

Método Map

Exemplo - método map()

Método Reduce

O **método reduce()** pode ser usado quando desejamos realizar alguma somatória ou então quando desejamos mesclar vários arrays em um único.

Esse método **reduz todos os valores de um array em um único resultado**, baseando-se na função que informamos para ele.

A função recebe dois parâmetros: o valor acumulado até o momento e o valor atual do elemento.

Método Reduce

Exemplo - método reduce()

Método Filter

O **metódo filter()**, como o próprio nome diz, ele tem o objetivo de filtrar as informações de um array.

Sua funcionalidade consiste em informarmos para ele uma condição. Ele irá aplicar essa condição em todos os itens de nosso array e aqueles que se enquadarem na condição serão retornados e adicionados ao novo array de saída.

Dessa forma, diferente do que ocorre no map() e no reduce(), o filter() irá retornar sempre true ou false.

Método Filter

Exemplo - método filter()

JavaScript Assíncrono

JavaSCript - Callbacks.

Callback é um tipo de função que só é executada após o processamento de outra função.

Na linguagem JavaScript, quando uma função é passada como um argumento de outra, ela é, então, chamada de callback.

JavaScript - Promises.

Em JavaScript, uma promise é um objeto que representa o resultado de uma operação assíncrona.

As promises são uma forma de gerenciar operações assíncronas de forma mais eficiente e segura.

As promises são criadas usando a função new Promise().

JavaScript - Promises.

Uma Promise é um objeto que possui três estados possíveis:

- **Pending (Pendente):** O estado inicial, quando a operação assíncrona está em andamento e ainda não foi concluída.
- **Fulfilled (Resolvida):** A operação assíncrona foi bem-sucedida, e a Promise retorna um valor (geralmente os resultados da operação).
- **Rejected (Rejeitada):** A operação assíncrona falhou, e a Promise retorna um motivo ou erro associado.

JavaScript - Promises.

JavaScript – Async/await

As funções assíncronas funcionam como Promises, porém com uma sintaxe mais simples.

O JavaScript async/await é uma maneira mais limpa e legível de lidar com código assíncrono em comparação com as Promises e os callbacks.

Ele foi introduzido na versão ECMAScript 2017 (ES8) para simplificar o tratamento de operações assíncronas, tornando o código mais parecido com código síncrono.

JavaScript – Async/await

Async: A palavra-chave **async** é usada para criar funções assíncronas. Funções assíncronas podem conter operações assíncronas e retornam uma Promise. Isso permite que você use **await** dentro da função para esperar que as Promises sejam resolvidas.

Await: A palavra-chave **await** é usada dentro de funções assíncronas para pausar a execução até que a Promise seja resolvida. Isso evita a pirâmide de callback (callback hell) que pode ocorrer com callbacks aninhados.

Referências:

Flanagan, D. (2012). JavaScript: O Guia Definitivo (6^a ed.). Bookman.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>