Introdução ao JavaScript

Seção 1: O que é JavaScript?

JavaScript é uma linguagem de programação de alto nível, interpretada e baseada em eventos. É amplamente utilizada para o desenvolvimento de aplicações web, tanto no lado do cliente (front-end) quanto no lado do servidor (back-end).

- **Características principais**:
  - **Interpretada**: Diferente de linguagens como C ou Java, JavaScript é interpretada diretamente pelo navegador ou ambiente de execução, sem a necessidade de uma compilação prévia.
  - **Dinamicamente tipada**: As variáveis podem mudar de tipo durante a execução.
  - **Baseada em eventos**: É comum utilizarmos eventos, como cliques de usuário, para disparar funções.

Seção 2: História do JavaScript

JavaScript foi criado por **Brendan Eich** em 1995, enquanto trabalhava na Netscape Communications Corporation. O objetivo inicial era adicionar interatividade às páginas web.

- **Marcos históricos**:
  - **1995**: JavaScript é criado em apenas 10 dias com o nome inicial de "Mocha".
  - **1996**: O nome é alterado para **JavaScript**, em uma tentativa de capitalizar sobre a popularidade da linguagem Java. Apesar do nome, as duas linguagens são bem diferentes.
  - **1997**: O ECMA International padroniza o JavaScript como **ECMAScript**.
  - **2009**: **Node.js** é lançado, permitindo a execução de JavaScript no lado do servidor.
  - **2015**: A versão ECMAScript 6 (ES6) introduz importantes atualizações, como `let`, `const`, arrow functions e promises.

Seção 3: Utilização no Front-End

No front-end, o JavaScript permite interagir com o **Document Object Model** (DOM) para criar páginas dinâmicas.

- **Exemplo de uso no Front-End**:

```
<button id="clickMe">Clique aqui</button>
<p id="text"></p>

<script>
  document.getElementById("clickMe").addEventListener("click", function() {
    document.getElementById("text").textContent = "Você clicou no botão!";
  });
</script>
```

Neste exemplo, ao clicar no botão, o JavaScript atualiza o conteúdo do elemento `<p>` para exibir a mensagem "Você clicou no botão!".

Seção 4: Utilização no Back-End com Node.js

Node.js é um ambiente de execução de JavaScript no lado do servidor, criado por **Ryan Dahl** em 2009. Ele foi construído sobre o motor V8 do Google Chrome, permitindo que os desenvolvedores utilizem JavaScript para construir aplicativos de back-end, como APIs e servidores.

- **História do Node.js**:
  - Criado para resolver o problema de desempenho com operações de I/O (Input/Output) em servidores tradicionais.
  - Node.js é **single-threaded**, ou seja, utiliza um único thread para gerenciar várias conexões, mas é altamente eficiente ao gerenciar operações de I/O através do modelo **assíncrono**.
  - Popularidade do **NPM** (Node Package Manager), que oferece mais de 1 milhão de pacotes prontos para uso.

Seção 5: Exemplo de Aplicação Back-End com Node.js

```
// Importa o módulo HTTP nativo do Node.js
const http = require('http');

// Cria um servidor que responde "Hello, World!" a todas as requisições
const server = http.createServer((req, res) => {
  res.statusCode = 200; // Define o status de sucesso
  res.setHeader('Content-Type', 'text/plain'); // Define o tipo de conteúdo
  res.end('Hello, World!\n'); // Envia a resposta
});

// Define a porta do servidor
const port = 3000;
server.listen(port, () => {
  console.log(`Servidor rodando em http://localhost:${port}/`);
});
```

- **Como funciona**:
  - O código utiliza o módulo `http` para criar um servidor que escuta na porta 3000.
  - Toda vez que o servidor recebe uma requisição, ele responde com "Hello, World!".

Para executar esse código:

1. Salve o arquivo como `app.js`.
2. No terminal, execute `node app.js`.
3. Acesse `http://localhost:3000/` no navegador, e você verá a mensagem "Hello, World!".

Seção 6: Diferenças Entre Front-End e Back-End em JavaScript

- **Front-End**:
  - Executado no navegador.
  - Responsável pela interação com o usuário e manipulação da interface gráfica.
  - Pode acessar o DOM diretamente.

- **Back-End**:
  - Executado no servidor, com Node.js.
  - Responsável por lidar com bancos de dados, autenticação e regras de negócios.
  - Não tem acesso direto ao DOM, mas pode manipular dados e responder a solicitações de clientes (como navegadores ou aplicativos móveis).

Controle de Fluxo, Funções, Objetos, Arrays e Programação Assíncrona com JavaScript

1. **Controle de Fluxo**

Condicionais: `if`, `else if`, `else`, `switch`

Exemplo: Verificação de idade para votação

```
let idade = 18;

if (idade < 16) {
  console.log("Você ainda não pode votar.");
} else if (idade >= 16 && idade < 18) {
  console.log("Você pode votar, mas não é obrigatório.");
} else {
  console.log("Você é obrigado a votar.");
}
```

Exemplo com `switch`:

```
let diaDaSemana = 3;

switch (diaDaSemana) {
  case 1:
    console.log("Domingo");
    break;
  case 2:
    console.log("Segunda-feira");
    break;
  case 3:
    console.log("Terça-feira");
    break;
  default:
    console.log("Dia inválido.");
}
```

Laços de Repetição: `for`, `while`, `do-while`

Exemplo com `for`: Contando até 5

```
for (let i = 1; i <= 5; i++) {
  console.log(i);
}
```

Exemplo com `while`: Imprimindo números até 3

```
let count = 1;

while (count <= 3) {
  console.log(count);
  count++;
}
```

Exemplo com `do-while`: Executando pelo menos uma vez

```
let number = 0;

do {
  console.log("Executando...");
  number++;
} while (number < 1);
```

---

2. **Funções**

Declaração de Funções

Exemplo: Função simples

```
function saudacao(nome) {
  return `Olá, ${nome}!`;
}

console.log(saudacao("Ana")); // Saída: Olá, Ana!
```

Funções como valores e callbacks

Exemplo de Função de Callback:

```
function processar(funcao, valor) {
  return funcao(valor);
}
```

```
function multiplicarPorDois(num) {
  return num * 2;
}

console.log(processar(multiplicarPorDois, 5)); // Saída: 10
```

Arrow Functions

Exemplo de Arrow Function:

```
const saudacaoArrow = (nome) => `Olá, ${nome}!`;

console.log(saudacaoArrow("Carlos")); // Saída: Olá, Carlos!
```

Escopo de Variáveis

Exemplo: Escopo Global e Local:

```
let nomeGlobal = "Pedro"; // Variável global

function mostrarNome() {
  let nomeLocal = "João"; // Variável local
  console.log(nomeLocal);
}

mostrarNome(); // Saída: João
console.log(nomeGlobal); // Saída: Pedro
```

---

3. **Objetos e Arrays**

Criação e Manipulação de Objetos

Exemplo: Criando um objeto carro

```
let carro = {
  marca: "Toyota",
  modelo: "Corolla",
  ano: 2020,
  mostrarDetalhes: function() {
    console.log(`Carro: ${this.marca} ${this.modelo}, Ano: ${this.ano}`);
  }
};

carro.mostrarDetalhes(); // Saída: Carro: Toyota Corolla, Ano: 2020
```

Arrays e Métodos de Iteração

Exemplo: Manipulando arrays

```
let numeros = [1, 2, 3, 4, 5];

// Usando forEach
numeros.forEach(num => console.log(num));

// Usando map
let numerosMultiplicados = numeros.map(num => num * 2);
console.log(numerosMultiplicados); // Saída: [2, 4, 6, 8, 10]

// Usando filter
let numerosMaioresQueTres = numeros.filter(num => num > 3);
console.log(numerosMaioresQueTres); // Saída: [4, 5]

// Usando reduce
let soma = numeros.reduce((total, num) => total + num, 0);
console.log(soma); // Saída: 15
```

---

4. **Programação Assíncrona**

Callbacks

Exemplo com `setTimeout` (Callback Simples)

```
function saudacao() {
  console.log("Olá, Mundo!");
}

console.log("Início");

setTimeout(saudacao, 2000); // Executa após 2 segundos

console.log("Fim");
```

Promises

Exemplo de Promise Simples

```
let promessa = new Promise((resolve, reject) => {
  let sucesso = true;

  if (sucesso) {
```

```
    resolve("A operação foi bem-sucedida!");
  } else {
    reject("Falha na operação.");
  }
});

promessa
  .then(mensagem => console.log(mensagem))
  .catch(erro => console.log(erro));
```

Async/Await

Exemplo com `async` e `await`

```
function esperar(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function saudacaoAssincrona() {
  console.log("Processando...");
  await esperar(2000);
  console.log("Olá, depois de 2 segundos!");
}

saudacaoAssincrona();
```