Curso de JavaScript 2023

Módulo 4 - Aula 08: JavaScript Moderno e ES6+

Instrutor: *Prof. Me. Mário Carvalho*
E-mail para contato: mario.carvalho@ufms.br

Realização: *UFMS e Semadur*

1. Sobre o curso

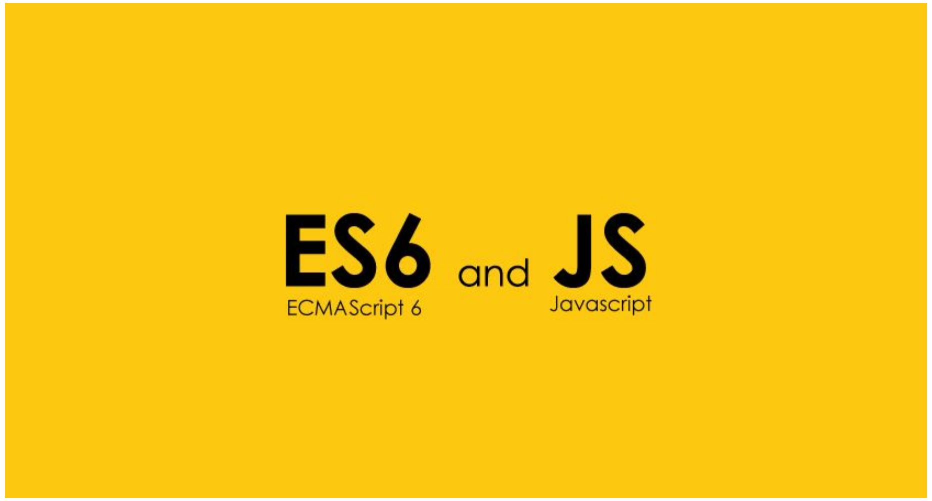
- Instrutor: **Prof. Me. Mário de Araújo Carvalho**
- **E-mail para contato:** mario.carvalho@ufms.br
- **Sala Google Meet: https://meet.google.com/fcq-djzs-dzd**
- Repositório oficial do curso:
- https://github.com/MarioCarvalhoBr/semadur-curso-javascript-2023
- **Ava:** https://www.eadfapec.com.br/course/view.php?id=245&section=4
- Instituição: **UFMS e Semadur**
- Modalidade: **Online síncrono**
- Duração: 45 horas

Introdução

- O que é ES6+?

Definição: ES6, também conhecido como ECMAScript 2015, é uma versão atualizada do JavaScript que trouxe novas funcionalidades e melhorias na linguagem.

Impacto: Modernizar o JavaScript, tornando-o mais eficiente, legível e fácil de manter.

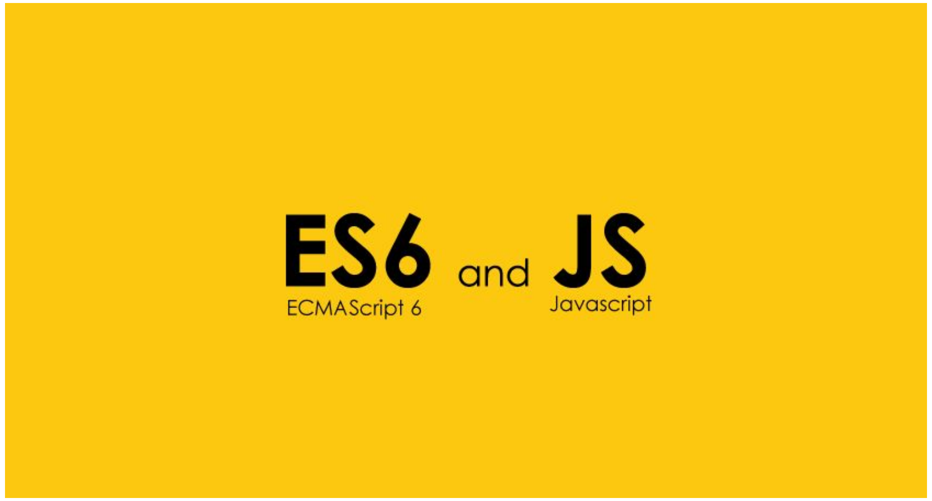


Introdução

- Consequências

Evolução do JavaScript: ES6+ introduziu melhorias significativas que são fundamentais no desenvolvimento moderno.

Continuando a Aprendizagem: Explore mais recursos e funcionalidades do ES6+ para aprimorar suas habilidades.

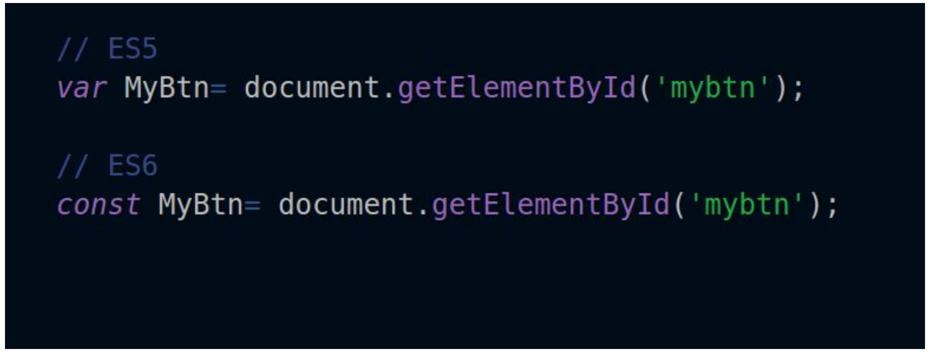


Introdução

- Novas Funcionalidades do ES6

Principais Adições em ES6:

- **Let e Const:** Novas palavras-chave para declaração de variáveis.
- **Arrow Functions:** Sintaxe concisa para funções.
- **Template Strings:** Facilita a criação de strings complexas.
- **Destructuring:** Desestruturação de objetos e arrays.



```
// ES5
var MyBtn= document.getElementById('mybtn');

// ES6
const MyBtn= document.getElementById('mybtn');
```

```
//ES6

const contacts={
    name:'said',
    famillyName:'Hayani',
    age:22
}

let{name,famillyName,age}=contacts
console.log(name)
console.log(famillyName)
console.log(age)
// output
// said
// Hayani
// 22
```

Introdução

- **Let e Const**

Declarando Variáveis Modernamente

Let: Variável com escopo de bloco, não hoistada.
Const: Constante de escopo de bloco, imutável.

Exemplo Prático

```
let idade = 30;
const NOME = "Alice";
```

Introdução

- **Arrow Functions**

Funções mais Enxutas

Sintaxe Reduzida: Não necessita da palavra-chave '***function***'.
***'this'* Contextual**: *this* é vinculado ao contexto onde a função foi escrita.

Exemplo Prático

```
// ES6 Arrow function
const minhaFuncao = nome => {
return `Olá ${nome}!`;
}
console.log(minhaFuncao("Mário"));
```

Introdução

- **Arrow Functions**

Funções mais Enxutas

Sintaxe Reduzida: Não necessita da palavra-chave '***function***'.
***'this'* Contextual**: *this* é vinculado ao contexto onde a função foi escrita.

Exemplo Prático:

```
// ES6 Arrow function
const minhaFuncao2 = (nome, idade) => {
   return `Olá ${nome}! Você tem ${idade}`;
}
console.log(minhaFuncao2("Mário", 25));
```

Introdução

- **Template Strings**

Construindo Strings com Facilidade

Interpolação: Inserir expressões diretamente nas strings.
Strings Multilinhas: Criar strings em várias linhas com facilidade.

Exemplo Prático:

```
let nome = "Ana";
let saudacao = `Olá, ${nome}!`;
```

https://www.freecodecamp.org/portuguese/news/javascript-es6-faca-mais-escrevendo-menos/

Introdução

- **Destructuring**

Desempacotando Dados com Elegância

Arrays e Objetos: Extraia dados de forma eficiente e legível.

Exemplo Prático

```
let [primeiro, segundo] = [1, 2];
let {nome, idade} = {nome: "Carlos", idade: 25};
console.log(primeiro)
console.log(idade)
```

Exercícios

Exercícios e Discussão

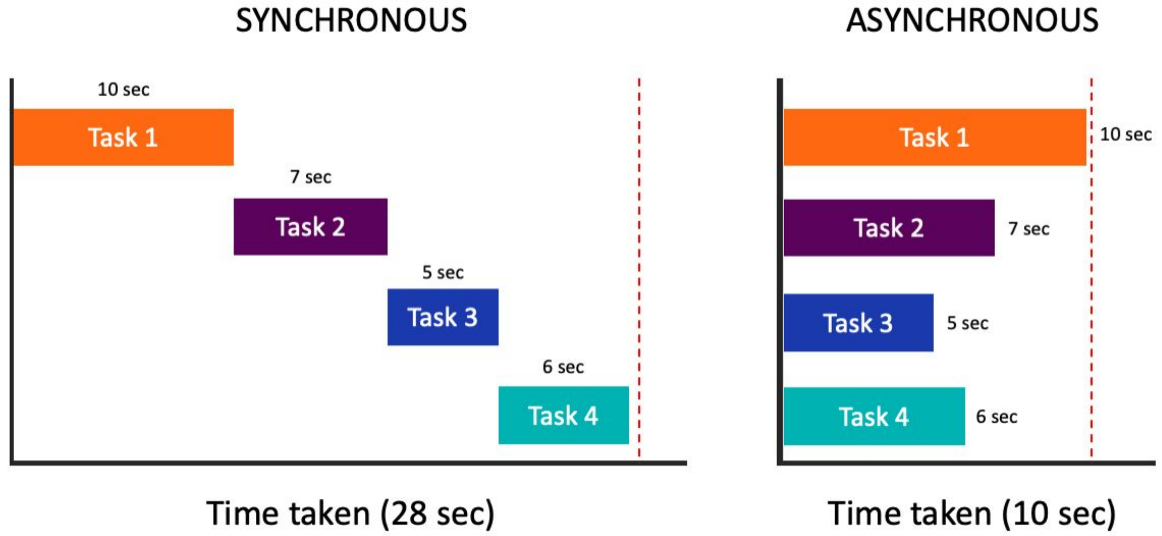
1. **Transformar Funções:** Converta funções tradicionais em arrow functions.
2. **Template Strings:** Crie uma string complexa usando template literals.
3. **Praticando Destructuring:** Extraia dados de um objeto complexo.

Promises, async/await

- Introdução à Programação Assíncrona
  - O que é Programação Assíncrona?

Definição: Execução de tarefas de forma que elas não bloqueiem a execução do restante do código.

Importância: Fundamental para operações como solicitações de rede, acesso a arquivos, etc.



Promises, async/await

Promises em JavaScript

- O que são Promises?

Definição: Um objeto que representa a eventual conclusão ou falha de uma operação assíncrona.
Estado: Uma Promise pode estar em um dos três estados: pendente, resolvida ou rejeitada.

Exemplo Prático

```
let promessa = new Promise((resolve, reject) => {
    // Operação assíncrona
});
```

Promises, async/await

Trabalhando com Promises

- Consumindo Promises

then(): Para manipular o resultado quando a Promise é resolvida.
catch(): Para capturar erros se a Promise for rejeitada.

Exemplo de código:

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

Promises, async/await

- **async/await**
  - **Simplificando Promises com async/await**

async: Declara uma função assíncrona.
await: Espera pela resolução de uma Promise dentro de uma função async.

Exemplo prático:

```
async function operacaoAssincrona() {
    try {
        let resultado = await promessa;
        console.log(resultado);
    } catch (erro) {
        console.error(erro);
    }
}
```

Promises, async/await

- **Práticas Comuns com async/await**
  - **Manipulação de Várias Promises**

Promise.all(): Aguarda a resolução de múltiplas Promises.

Exemplo prático:

```
async function carregarDados() {
    let [resultado1, resultado2] = await
Promise.all([promessa1, promessa2]);
       // ...
}
```

Exercícios

Exercícios e Discussão

1. **Criar e Consumir uma Promise:** Implemente uma Promise simples e consuma-a com then e catch.
2. **Função Assíncrona com await:** Crie uma função async que utiliza await para esperar uma Promise.
3. **Manipulação de Múltiplas Promises:** Use Promise.all para lidar com várias Promises simultaneamente.

Promises, async/await

Conclusões

- **Evolução da Programação Assíncrona:** Promises e async/await tornaram a programação assíncrona em JavaScript mais acessível e compreensível.

- **Prática Contínua:** Aprofunde seus conhecimentos através de mais prática e exploração de casos de uso avançados.

Recursos JavaScript Moderno

- **Arrow Functions, Template Literals, Spread/Rest Operators**

  - **Visão Geral**

Objetivo: Explorar as funcionalidades modernas do JavaScript que tornam a linguagem mais expressiva e eficiente.
Tópicos: Arrow Functions, Template Literals, Spread/Rest Operators.

Arrow Functions

O que são Arrow Functions?

- **Manipulação de Várias Promises**

Definição: Uma forma mais curta de escrever funções em JavaScript.

Características: Não possui seu próprio this, não adequada para métodos, não pode ser usada como construtoras.

Exemplo prático:

```
const somar = (a, b) => a + b;
console.log(somar(2, 3)); // Saída: 5
```

Arrow Functions

- **Exercícios com Arrow Functions**

  - **Prática de Arrow Functions**

1. **Conversão de Funções:** Converta uma função tradicional em uma arrow function.
2. **Arrow Functions com Múltiplos Argumentos:** Escreva uma arrow function que aceita vários argumentos e os processa.

Template Literals

O que são Template Literals?

Definição: Permite a criação de strings complexas de forma mais legível e flexível.
Uso: Interpolação de expressões, criação de strings multilinhas.

Exemplo prático:

```
let nome = "Ana";
let saudacao = `Olá, ${nome}!`;
console.log(saudacao); // Saída: Olá, Ana!
```

Template Literals

Exercícios com Template Literals

1. **Interpolação de Variáveis:** Crie uma string complexa usando várias variáveis.
2. **Strings Multilinhas:** Utilize template literals para formar uma string em várias linhas.

Spread/Rest Operators

- **O que são Spread e Rest Operators?**

Spread Operator: Expande elementos de um iterável (como arrays) em lugares onde múltiplos argumentos/elementos são esperados.

Rest Operator: Agrupa argumentos em um array dentro de uma função.

Spread/Rest Operators

- **Exemplo Prático: Spread Operator**

```
let numeros = [1, 2, 3];
let maisNumeros = [0, ...numeros, 4];
console.log(maisNumeros);
// Saída: [0, 1, 2, 3, 4]
```

Spread/Rest Operators

- **Exemplo Prático: Rest Operator**

```
function sumarizar(...numeros) {
    return numeros.reduce((acc, num) =>
acc + num, 0);
}
console.log(sumarizar(1, 2, 3)); //
Saída: 6
```

Spread/Rest Operators

- **Exercícios com Spread/Rest Operators**

1. **Combinação de Arrays:** Use o spread operator para combinar arrays.
2. **Função com Número Variável de Argumentos:** Utilize o rest operator em uma função para lidar com um número variável de argumentos.

Arrow Functions, Template Literals, Spread/Rest Operators

Conclusões

- **Potencializando o JavaScript:** As funcionalidades modernas do JavaScript como arrow functions, template literals e spread/rest operators oferecem uma sintaxe mais enxuta e poderosa.

- **Prática Continuada:** Explore estas funcionalidades em diferentes cenários para aprimorar suas habilidades de codificação.

Módulos ES6 export/import

- **Introdução aos Módulos ES6**
  - **O que são Módulos ES6?**

Definição: Módulos ES6 permitem a separação de códigos em diferentes arquivos, cada um encapsulando suas próprias funcionalidades.
Vantagens: Facilita a manutenção, reutilização de código e dependências claras.

Exportando em Módulos ES6

Como Exportar Código?

Export: A palavra-chave ***export*** é usada para exportar funções, objetos ou primitivas de um módulo.

Exemplo prático:

```
// arquivo: mathUtils.js
export function somar(a, b) {
    return a + b;
}

export const PI = 3.14159;
```

Importando em Módulos ES6

- **Como Importar Código?**

Export: A palavra-chave ***import*** é usada para trazer código exportado de outro módulo.

Exemplo prático:

```
// arquivo: app.js
import { somar, PI } from './mathUtils.js';

console.log(somar(2, 3)); // Saída: 5
console.log(PI); // Saída: 3.14159
```

Exportações e Importações Avançadas

- **Exportações Nomeadas e Padrão**

Exportações Nomeadas: Exporta múltiplos valores. Ao importar, deve-se usar os mesmos nomes.

Exportação Padrão: Cada módulo pode ter um export default, que é importado sem chaves.

Exemplo prático:

```
// arquivo: defaultMathUtils.js
export default function multiplicar(a, b) {
    return a * b;
}
```

Importação Dinâmica de Módulos

Uso de Importações Dinâmicas

Importações Dinâmicas: Usar import() para carregar módulos de forma assíncrona.

Exemplo prático:

```
// Usando importação dinâmica
import('./mathUtils.js').then(({ somar }) => {
    console.log(somar(5, 10)); // Saída: 15
});
```

Exercícios

Exercícios e Discussão

1. **Criar e Exportar um Módulo:** Crie um módulo com várias funções e exporte-as.
2. **Utilizar Exportações Nomeadas e Padrão:** Crie um módulo que use ambos os tipos de exportação e importe-os em outro arquivo.

Módulos ES6

Conclusões

- **Modularidade em JavaScript:** O sistema de módulos ES6 introduz uma forma poderosa de organizar e gerenciar dependências em aplicações JavaScript.
- **Importância da Prática:** Aprofunde seu entendimento praticando a criação de módulos e explorando diferentes formas de exportação e importação.

Encerramento

- \- Resumo dos tópicos abordados e orientações sobre recursos para continuar aprendendo JavaScript:
  - MDN Web Docs: **https://developer.mozilla.org/pt-BR/docs/Web/JavaScript**

- \- Próximos passos e recursos adicionais para aprendizado autônomo:
  - HTML: https://www.w3schools.com/html/default.asp
  - JavaScript: https://www.w3schools.com/js/default.asp

Dúvidas?

Obrigado!

Mário Carvalho
mario.carvalho@ufms.br