JavaScript

Túlio Braga Alves
UNISAGRADO
Ensino Superior de Excelência

MiniCurso Centro Universitário Sagrado Coração

Conteúdo

Módulo 1: Introdução ao JavaScript

- História e importância do JS;
- Instalação do VS *Code;*
- Introdução ao console do navegador.

Módulo 2: Variáveis e Operações básicas

- *Var, let e const*, boas práticas;
- Operações aritméticas;
- Operadores de comparação, concatenação.

Módulo 3: Manipulação do DOM

- O que é o DOM;
- Métodos de busca de elementos;
- Alteração de conteúdo;
- Manipulação de eventos.

Módulo 4: Funções

- Funções tradicionais e arrow functions;
- Parâmetros e retorno.

Módulo 5: Manipulação de Objetos

- Criando e manipulando objetos;
- Arrays de objetos;
- Extraindo valores de objetos.

Módulo 6: Funções assíncronas

- Introdução a assincronia;
- Trabalhando com API's;
- Try, Fetch, Catch.

O que abordaremos no primeiro dia?

Iremos abordar alguns conteúdos iniciais sobre JavaScript. Aprenderemos conceitos básicos, como variáveis, tipos de dados, operadores e estruturas condicionais, construindo uma base sólida para o segundo dia. Nesse segundo momento, aprofundaremos em boas práticas e estruturas amplamente usadas no mercado de trabalho, como loops, funções e manipulação do DOM, essenciais para desenvolver projetos profissionais e escaláveis.

Olá!

Meu nome é Túlio, tenho 18 anos e sempre fui apaixonado em computadores e principalmente, sair fuçando em tudo relacionado a isso.

Trabalhei desde os meus 13 anos editando vídeos como freelancer, até ser contratado por uma agência de audiovisual. No final de 2023 comecei a estudar programação por conta própria, e entrei na faculdade logo em seguida.

Em julho de 2024 consegui um estágio de desenvolvimento em uma empresa em Bauru, na qual eu trabalhava com JavaScript, Java e SQL, até ser contratado como desenvolvedor Web em Piratininga.

/tulioalves7 /tuliobragaalves7 /tulioalvess7

UNISAGRADO
Ensino Superior de Excelência

Por que **JavaScript**?

1. JavaScript é a 4ª linguagem mais usada no mundo: Sua popularidade global garante uma vasta comunidade e muitas oportunidades de trabalho.

2. Versatilidade full-stack: Com JavaScript, você pode desenvolver tanto no front-end quanto no back-end, permitindo criar aplicações completas com uma única linguagem.

3. Alta demanda no mercado: Empresas de todos os portes buscam desenvolvedores JavaScript, tornando-o uma habilidade essencial para quem deseja ingressar ou crescer na área de tecnologia.

UNISAGRADO
Ensino Superior de Excelência

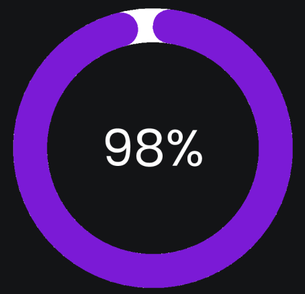
Versatilidade no **Front-end** e **Back-end**

- **No Front-end**
- JavaScript permite construir interfaces dinâmicas e interativas para o usuário final, dando vida ao HTML e CSS. Com frameworks como React, Angular, e Vue, é possível criar Single Page Applications (SPAs), onde a navegação é rápida e o conteúdo carrega sem a necessidade de recarregar a página. Além disso, manipula o DOM e permite criar animações, formulários interativos, e notificações.

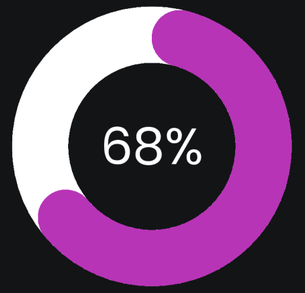
- **No Back-end**
- Com o surgimento do Node.js, o JavaScript ganhou ainda mais força, expandindo sua atuação para o servidor. No Back-end, ele permite:
- Processar requisições e respostas de servidores web.
- Gerenciar bancos de dados com bibliotecas como Mongoose para MongoDB.
- Criar APIs RESTful e GraphQL, permitindo uma comunicação eficiente com o front-end.
- Trabalhar com autenticação de usuários e segurança de dados.

Com JavaScript, você constrói do começo ao fim.

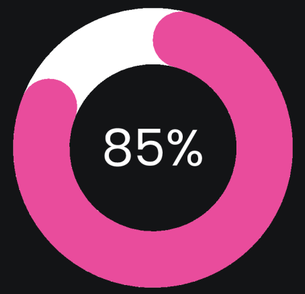
Os **números** não **mentem**!



98% das páginas da web utilizam JavaScript de alguma forma, tornando-o essencial para o desenvolvimento web moderno.



Dos desenvolvedores full-stack utilizam JavaScript tanto no front-end quanto no back-end (com Node.js), devido à sua versatilidade.



85% das empresas de tecnologia citam JavaScript como uma das linguagens mais procuradas em seus processos de contratação.

Módulo 1 - Introdução ao JavaScript

01

Como Surgiu o JavaScript?

O JavaScript surgiu em 1995, criado por Brendan Eich enquanto trabalhava na Netscape. O objetivo era tornar as páginas da web mais interativas, já que, até então, elas eram principalmente estáticas. A linguagem foi projetada para ser simples e permitir a interação entre o usuário e o navegador, complementando o HTML e o CSS.

E assim, nasceu o **JavaScript**, para dar vida à web.

Ascensão do JS

JavaScript inicialmente foi visto como uma linguagem "de apoio" comparada a outras como Java ou C++, mas com o tempo, o próprio navegador virou a "plataforma" e JS tomou conta.
Hoje em dia, o JS alimenta grandes frameworks, como React, Angular e Vue.

O que era apenas um **coadjuvante** tornou-se o protagonista da **web**.

02

03

A Padronização do JavaScript

Em 1997, o JavaScript foi padronizado pela ECMA International, sob o nome ECMAScript. Isso garantiu que a linguagem fosse adotada de maneira uniforme por diferentes navegadores, evitando incompatibilidades. A versão ECMAScript 3, lançada em 1999, foi uma das versões mais importantes, estabelecendo as bases para o crescimento do JavaScript.

Com a padronização, o **JavaScript** ganhou **asas** para **voar** mais longe.

A Era do Ajax

Nos anos 2000, o AJAX (Asynchronous JavaScript and XML) revolucionou a web. Com ele, foi possível criar páginas dinâmicas que atualizam conteúdo sem precisar recarregar toda a página. Esse avanço trouxe uma nova fase de interatividade, impulsionando o desenvolvimento de aplicações web mais rápidas e responsivas, como Gmail e Google Maps.

Com o AJAX, a web deixou de ser estática e começou a respirar.

04

05

V8 e o Avanço do JavaScript

Em 2008, o Google lançou o motor V8 JavaScript Engine, que melhorou drasticamente a performance do JavaScript nos navegadores, permitindo o uso da linguagem em aplicativos mais complexos e rápidos. Isso abriu caminho para o JavaScript não apenas no frontend, mas também no backend com o surgimento do Node.js.

O V8 deu ao JavaScript a velocidade necessária para dominar toda a web, do frontend ao backend.

O JavaScript Moderno

Com o lançamento do ECMAScript 6 (ES6) em 2015, o JavaScript ganhou uma série de novos recursos, como classes, arrow functions, let/const, e template literals. Essas atualizações transformaram a linguagem em algo mais robusto e orientado a objetos, consolidando sua posição como uma das linguagens mais usadas no mundo, tanto no frontend quanto no backend.

Com tantas vantagens, o **VS Code** se tornou o companheiro favorito dos desenvolvedores **JavaScript.**

06

Biblioteca **VS** Framework

- **React**
- Um conjunto de funcionalidades reutilizáveis.
- Você controla o fluxo da aplicação e como usar as ferramentas.
- React oferece componentes e hooks para interfaces de usuário.

- **Angular, Next.JS**
- Estrutura completa que define um padrão arquitetônico.
- O framework controla o fluxo de execução do código.
- Angular e Next.js oferecem convenções e ferramentas integradas.

- **Biblioteca:** você decide como e quando usar suas funções.
- **Framework:** ele dita a organização e fluxo da aplicação.

UNISAGRADO
Ensino Superior de Excelência

Porque usar o **VS Code**?

Além de ser um editor leve e gratuito, ele possui uma imensa biblioteca de extensões que facilitam o desenvolvimento em JavaScript. É o editor de texto preferido de 80% dos desenvolvedores JS.

Com o ES6, o **JavaScript** deu um salto evolutivo, tornando-se mais poderoso e versátil.

03

UNISAGRADO
Ensino Superior de Excelência

Módulo 2 - Declaração de Variáveis e Operações básicas

var, let, const.

var

Foi a única forma de declarar variáveis por muito tempo, mas seu escopo global pode gerar problemas. E por isso caiu em desuso.

let

Escopo de bloco, mais seguro e previsível. Exemplo: Dentro de loops, *let* é a escolha ideal.

const

Para valores que nunca devem mudar. É ótimo para criar variáveis constantes, como URLs de APIs.

Escolher a variável certa faz toda a diferença no código.

UNISAGRADO
Ensino Superior de Excelência

var VS let

```
1  if (true) {
2      var x = 10;
3    }
4    console.log(x);  // Saída: 10 (x fica acessível fora do bloco 'if')
```

Mesmo declarando x dentro do bloco if, ela fica acessível fora dele, porque var não respeita o escopo de bloco, apenas escopo de função. É como se você deixasse a porta de um quarto aberta e tudo que fosse declarado lá dentro pudesse sair para o corredor. Já com let ou const, a porta se fecha, limitando o acesso ao conteúdo do quarto.

Aqui, a variável y foi declarada com let dentro do bloco if, o que faz com que ela só exista nesse escopo específico. Ao tentar acessá-la fora do bloco, ocorre um erro de referência, pois let respeita o escopo de bloco, diferente de var. É como manter a variável dentro de uma sala trancada, acessível apenas enquanto você está dentro dessa sala.

```
1  if (true) {
2    let y = 20;
3  }
4  console.log(y);  // ReferenceError: y is not defined
```

Enquanto var deixa tudo escapar, let e const mantêm tudo no seu lugar.

UNISAGRADO
Ensino Superior de Excelência

Operações Aritméticas

No JavaScript, realizar operações matemáticas é direto e fácil. Você pode fazer adição, subtração, multiplicação e divisão diretamente com variáveis.

```javascript
let a = 10;
let b = 5;
console.log(a + b); // 15 (Adição)
console.log(a - b); // 5 (Subtração)
console.log(a * b); // 50 (Multiplicação)
console.log(a / b); // 2 (Divisão)
```

Isso nos dá uma liberdade gigante para realizarmos cálculos, médias, e desenvolver programas com múltiplas funcionalidades,

Operações Aritméticas

No JavaScript, você pode facilmente calcular a média das notas de um aluno usando operações aritméticas. Basta somar as notas e dividir pela quantidade de avaliações.

```javascript
let nota1 = 8;
let nota2 = 7.5;
let nota3 = 9;
let media = (nota1 + nota2 + nota3) / 3;
console.log(media); // 8.17 (Média)
```

Neste exemplo, calculamos a média de três notas e exibimos o resultado diretamente no console.

UNISAGRADO
Ensino Superior de Excelência

Estruturas Condicionais

O JavaScript usa if e else para tomar decisões com base em condições. O if executa o bloco de código quando a condição é verdadeira. O else é executado quando a condição não é atendida, permitindo dois caminhos possíveis.

Exemplo: Verificando se o pedido de um cliente atende ao valor mínimo para aplicar um desconto.

```
1  let valorPedido = 150;
2  if (valorPedido ≥ 100) {
3    console.log("Desconto aplicado!");
4  } else {
5    console.log("Pedido abaixo do valor mínimo para desconto.");
6  }
7
```

Com if e else, o **JavaScript** decide qual caminho seguir.

UNISAGRADO
Ensino Superior de Excelência

Estruturas Condicionais

Para mais de duas condições, podemos usar else if. Assim, o JavaScript verifica cada condição em sequência até encontrar uma verdadeira, executando o bloco correspondente. Se nenhuma condição for verdadeira, o else final é executado.

Exemplo: Determinando a faixa de preço de um produto com base no seu valor.

```
1  let precoProduto = 250;
2
3  if (precoProduto > 500) {
4    console.log("Produto de alto custo");
5  } else if (precoProduto ≥ 200) {
6    console.log("Produto de custo médio");
7  } else {
8    console.log("Produto acessível");
9  }
```

Com else if, o **JavaScript** avalia todas as possibilidades.

Exemplo Prático: if e else

Imagine que queremos verificar a idade de um cliente para determinar se ele pode acessar uma seção restrita de um site de e-commerce. Usamos if para permitir acesso se a idade for maior que 18, e else para restringir.

```
1  let idadeCliente = 20;
2  if (idadeCliente ≥ 18) {
3    console.log("Acesso à seção restrita permitido");
4  } else {
5    console.log("Acesso negado à seção restrita");
6  }
```

Com if e else, o código responde conforme as condições.

Comparação | == VS ===

O operador == compara apenas os valores, realizando conversões de tipo se necessário, enquanto o === compara tanto os valores quanto os tipos, sem conversões. Isso significa que, com ==, uma string '123' pode ser igual ao número 123, mas com ===, a comparação falha porque os tipos são diferentes. Use === para garantir que tanto o valor quanto o tipo sejam idênticos, evitando resultados inesperados.

```
let userID = 123; // Número armazenado
let inputID = '123'; // Entrada do usuário como string (texto)

if (userID == inputID) {
  console.log('Login com sucesso!'); // Isso retornaria true, mas o tipo é diferente.
}

if (userID === inputID) {
  console.log('Login com sucesso!'); // Isso retornaria false, pois o tipo é diferente.
}
```

Com ===, você garante que valor e tipo estão realmente alinhados.

Loops

O while executa um bloco de código enquanto uma condição for verdadeira. É ideal quando não sabemos exatamente quantas vezes o código deve repetir.

Exemplo: Processando pedidos enquanto houver estoque disponível.

```
let estoque = 5;

while (estoque > 0) {
  console.log("Pedido processado!");
  estoque--;
}
```

Com while, o JavaScript continua enquanto a condição é válida.

UNISAGRADO
Ensino Superior de Excelência

Loops

O do...while é semelhante ao while, mas a condição é verificada após a execução do bloco, garantindo que o código dentro do do seja executado ao menos uma vez.

Exemplo: Confirmando a tentativa de pagamento de um cliente até que seja concluída.

```
let tentativas = 0;
let pagamentoConcluido = false;

do {
  console.log("Tentando processar pagamento ... ");
  tentativas++;
  pagamentoConcluido = (tentativas === 3);
} while (!pagamentoConcluido);

```

O do...while garante pelo menos uma execução antes da verificação.

UNISAGRADO
Ensino Superior de Excelência

Loops

O for é ideal para repetições com um número conhecido de vezes. Ele inclui uma inicialização, uma condição, e uma atualização em uma única linha.

Exemplo: Enviando uma notificação para todos os 10 novos clientes cadastrados.

```
for (let i = 1; i ≤ 10; i++) {
    console.log(`Notificação enviada para o cliente número ${i}`);
  }

```

Com for, você controla cada passo da repetição.

Concatenação de **Strings**

Antes do ES6, a forma mais comum de concatenar strings era utilizando o operador +. Esse método junta valores de strings ou combina strings com variáveis de uma forma simples.

```
1  let nome = "Tulio";
2  let saudacao = "Olá, " + nome + "! Bem-vindo.";
3  console.log(saudacao); // Olá, Tulio! Bem-vindo.
```

Mas isso pode acabar gerando problemas no futuro.

Concatenação de **Strings**

Usar + para concatenar strings pode facilmente levar a problemas, especialmente se você estiver misturando tipos de dados, ou diversas variáveis em conjunto.

```
1  let nome = "Tulio";
2  let idade = 18;
3  let profissao = "Desenvolvedor";
4  let curso = "Ciência da Computação";
5  let saudacao = "Olá, " + nome + "! Bem-vindo. Você tem " + idade + "Anos, sua profissão é " + profissao + " e você cursa " + curso + ". ";
6  console.log(saudacao); // Olá, Tulio! Bem-vindo. Você tem 18 Anos, sua profissão é Desenvolvedor e você cursa Ciência da Computação.
7
```

Neste exemplo, variáveis como nome, idade, e profissão são combinadas, mas essa abordagem pode gerar código confuso e difícil de manter. A solução moderna e mais eficiente é utilizar **Template Strings**, introduzidas no ES6, que tornam o código mais legível e simples.

UNISAGRADO
Ensino Superior de Excelência

Concatenação de **Strings**

Aqui, não precisamos do + para concatenar. Basta envolver a variável com ${} e inseri-la diretamente na string. Além disso, as Template Strings suportam múltiplas linhas, o que é muito útil para criar textos longos.
Observação: Ao invés de aspas (“), devemos usar (``) para interpolação de strings

```
let nome = "Tulio";
let idade = 18;
let profissao = "Desenvolvedor";
let curso = "Ciência da Computação";
let saudacao = `Olá, ${nome}! Bem-vindo. Você tem ${idade} Anos, sua profissão é ${profissao} e você cursa ${curso}`
console.log(saudacao); // Olá, Tulio! Bem-vindo. Você tem 18 Anos, sua profissão é Desenvolvedor e você cursa Ciência da Computação.
```

Módulo 3 - Manipulação do **DOM**

Por que surgiu o **DOM**?

1

- O HTML, sozinho, era estático. Não permitia mudanças após o carregamento da página.
- A necessidade de criar interações dinâmicas exigiu uma forma de manipular a página em tempo real.

2

- Antes do DOM, não havia um padrão claro para separar o conteúdo (HTML) do comportamento (JS).
- O DOM surgiu para fornecer uma interface clara entre o documento e o JavaScript.

3

- Para modificar a estrutura, estilo e conteúdo da página, o JavaScript precisava de uma "porta" para interagir com os elementos.
- O DOM fornece essa estrutura, permitindo modificar os elementos sem recarregar a página.

O DOM abriu a porta para um **HTML** dinâmico e interativo.

O que é **DOM**?

1. O DOM é uma interface que permite ao **JavaScript** acessar e manipular elementos e estruturas do HTML de uma página web.

2. Com o DOM, o JS pode modificar, adicionar ou remover elementos da página, alterando conteúdo e estilos em tempo real.

3. O HTML é o esqueleto da página, enquanto o **JavaScript** são os músculos que movem tudo por meio do DOM, criando interatividade.

O DOM é a ponte que conecta o **HTML** ao poder do **JavaScript**.

Por que buscar elementos no **DOM**?

1. **Adicionar interatividade:** Capturar cliques em botões, fazer validações de formulário, ou criar animações.

2. **Alterar conteúdo e estilos:** Atualizar textos, mudar cores, ou aplicar classes CSS em tempo real.

3. **Criar e remover elementos:** Adicionar novos itens (como em um chat) ou remover elementos sem recarregar a página.

Como acessamos algum **elemento web**?

- Busca por ID: O método mais rápido para encontrar um elemento único. getElementById()

```
let elemento = document.getElementById("meuId");
```

- Busca por Classe: Retorna todos os elementos que possuem uma determinada classe. getElementsByClassName()

```
let elementos = document.getElementsByClassName("minhaClasse");
```

Os métodos clássicos, como getElementById(), são menos usados hoje porque são limitados. Métodos modernos, como querySelector(), são mais flexíveis, permitindo buscas com seletores CSS e facilitando o trabalho em projetos maiores.

UNISAGRADO
Ensino Superior de Excelência

Métodos **Modernos** de Busca

- Busca por Seletores CSS: Encontra o primeiro elemento que combina com o seletor (ID, classe, tag). querySelector()

```
let elemento = document.querySelector("#botao");
```

- Busca por Seletores CSS: Retorna todos os elementos que combinam com o seletor. querySelectorAll()

```
let elementos = document.querySelectorAll(".minhaClasse");
```

Os métodos querySelector() e querySelectorAll() são amplamente usados por sua flexibilidade. Eles permitem buscar elementos com seletores CSS, dando mais precisão e controle. Por serem versáteis, tornaram-se a escolha padrão em projetos modernos, facilitando o desenvolvimento e a manutenção, especialmente em aplicações maiores.

UNISAGRADO
Ensino Superior de Excelência

Alteração de Conteúdo no **DOM**

innerHTML:

- Altera o conteúdo HTML de um elemento.
- Pode inserir código HTML.
- Risco: pode ser explorado por scripts maliciosos (cross-site scripting - XSS).

textContent:

- Altera apenas o texto.
- Mais seguro, pois não executa HTML

```
1  document.getElementById("meuTitulo").textContent = "Novo Título";
```

UNISAGRADO
Ensino Superior de Excelência

Manipulação de Eventos no **JavaScript**

Por que usar eventos?

- Eventos permitem interações dinâmicas.
- Reagem a ações do usuário, como cliques, teclas pressionadas, envio de formulários, etc.

addEventListener:

- Adiciona uma função que é chamada quando um evento específico ocorre.
- Muito flexível: permite associar múltiplos eventos ao mesmo elemento.

```
1  document.getElementById("meuBotao").addEventListener("click", function() {
2    alert("Botão foi clicado!");
3  });
```

UNISAGRADO
Ensino Superior de Excelência

Manipulação de Elementos no **DOM**

O DOM permite acessar e manipular elementos HTML com JavaScript, facilitando o desenvolvimento de páginas dinâmicas. Além dos métodos de busca como getElementById e querySelector, podemos modificar diretamente propriedades dos elementos, alterando estilos, conteúdos e atributos.

Exemplo: Mudando o estilo de um botão:

```
let botao = document.getElementById("meuBotao");
botao.style.backgroundColor = "blue";
botao.style.fontSize = "16px";

```

Com o DOM, o JavaScript dá vida ao HTML.

UNISAGRADO
Ensino Superior de Excelência

Eventos no **DOM**

Eventos são ações do usuário (como cliques e teclas pressionadas) ou do navegador (como o carregamento da página) que disparam uma resposta no JavaScript. Podemos “ouvir” eventos com métodos como addEventListener, tornando as páginas interativas e responsivas.

Exemplo: Detectando um clique no botão:

```
botao.addEventListener("click", () => {
    alert("Botão clicado!");
  });

```

Os eventos conectam o usuário com a página.

UNISAGRADO
Ensino Superior de Excelência

Eventos Comuns do Navegador

O JavaScript pode capturar eventos comuns como *click*, *mouseover*, *keyup*, *submit*, entre outros. Cada evento traz informações sobre a ação e permite respostas personalizadas para o usuário.

Exemplo: Mudando o conteúdo ao passar o mouse:

```javascript
let texto = document.getElementById("meuTexto");
texto.addEventListener("mouseover", () => {
  texto.textContent = "Você passou o mouse!";
});

```

Cada ação do usuário é uma oportunidade para interagir.

UNISAGRADO
Ensino Superior de Excelência

Eventos de **Carregamento**

Os eventos de carregamento, como DOMContentLoaded e load, garantem que o JavaScript execute apenas quando a página ou os recursos (imagens, vídeos) estejam prontos. Isso evita erros de carregamento de elementos.

Exemplo: Executando um código ao carregar o DOM:

```
1  document.addEventListener("DOMContentLoaded", () ⇒ {
2      console.log("O DOM foi carregado!");
3    });
4
```

Carregar na hora certa evita problemas na execução.

UNISAGRADO
Ensino Superior de Excelência

Modificando Atributos no DOM

Podemos modificar atributos HTML diretamente pelo JavaScript. Métodos como setAttribute e removeAttribute ajudam a ajustar dinamicamente as características de cada elemento na página.

Exemplo: Modificando o src de uma imagem:

```
1  let imagem = document.querySelector("img");
2  imagem.setAttribute("src", "novaImagem.jpg");
3
```

Os atributos mudam, a interação cresce.

UNISAGRADO
Ensino Superior de Excelência

Removendo Elementos do DOM

Para criar uma página dinâmica, às vezes é necessário remover elementos do DOM. Os métodos removeChild e remove são úteis para deletar elementos de maneira fácil e eficiente, permitindo um controle maior sobre o layout.

Exemplo: Removendo um item da lista ao clicar:

```
let item = document.getElementById("itemParaRemover");
item.remove();

```

Remover elementos limpa a página e melhora a experiência do usuário.

UNISAGRADO
Ensino Superior de Excelência

Módulo 4 - Funções

Funções

Funções são como pequenas “fábricas” no seu código: elas pegam entradas (parâmetros), processam algo e retornam um resultado.

Elas ajudam a organizar o código, evitam repetições e deixam o trabalho mais fácil.

Exemplo rápido: Imagine uma função “fazerSuco(fruta)” que recebe o nome de uma fruta e devolve um suco. Facilita, né?

Funções

A sintaxe é assim: function nome(param) { código }

```
1  function dizerOi(nome) {
2      return "Oi, " + nome + "!";
3  }
```

Funções tradicionais são versáteis e compatíveis com qualquer navegador, mas o código pode ficar um pouco extenso.

UNISAGRADO
Ensino Superior de Excelência

O Que São Arrow Functions?

Arrow functions (=>) vieram para encurtar o código, eliminando o function e usando uma “flecha”.

Sintaxe rápida: (param) => { código }

Elas também não têm o próprio this, então são ótimas quando você não quer que o valor de this mude.

UNISAGRADO
Ensino Superior de Excelência

Quando Usar Cada Tipo

Funções tradicionais: boas para código complexo e uso de this.

Arrow functions: ideais para callbacks (funções que passam para outras funções) e simplificar o código.

```
1  let dobrar = num => num * 2;
2  console.log(dobrar(4)); // Resultado: 8
```

UNISAGRADO
Ensino Superior de Excelência

Parâmetros em Funções

Parâmetros são como entradas que damos para as funções.

Eles permitem personalizar o que a função faz.

```
function saudar(nome, saudacao) {
    return saudacao + ", " + nome + "!";
}
```

UNISAGRADO
Ensino Superior de Excelência

Funções com Retorno

Retorno é o que a função “devolve” depois de ser executada.

```
function somar(a, b) {
    return a + b;
}
console.log(somar(3, 5)); // Resultado: 8
```

UNISAGRADO
Ensino Superior de Excelência

Resumo e Boas Práticas

Funções tradicionais são mais detalhadas e versáteis; arrow functions são compactas e ótimas para callbacks.

Use parâmetros e retorno para tornar funções flexíveis.

Dica: escolha o tipo de função que deixa seu código mais limpo e fácil de entender.

Módulo 5: Objetos e Arrays de Objetos

Introdução aos Objetos

Objetos são coleções de propriedades (valores com nomes) que representam características de algo.

Imagine um objeto “carro”: ele pode ter propriedades como cor, marca, modelo, etc.

```
let carro = {
    cor: "vermelho",
    marca: "Toyota",
    modelo: "Corolla"
};
```

UNISAGRADO
Ensino Superior de Excelência

Criando Objetos

Para criar um objeto, usamos { chave: valor }.

```
let pessoa = { nome: "Tulio", idade: 25 };
console.log(pessoa.nome); // Resultado: "Tulio"
```

UNISAGRADO
Ensino Superior de Excelência

Manipulando Objetos

Podemos adicionar, atualizar ou remover propriedades de um objeto.

```
let animal = { tipo: "cachorro", nome: "Bart" };
animal.idade = 3; // Adicionando nova propriedade
delete animal.nome; // Removendo propriedade
```

UNISAGRADO
Ensino Superior de Excelência

Arrays de Objetos

Arrays de objetos são listas de objetos. Úteis para representar múltiplos elementos com as mesmas características.

```
let turma = [
    { nome: "Tulio", idade: 20 },
    { nome: "Julia", idade: 18 }
];
```

UNISAGRADO
Ensino Superior de Excelência

Percorrendo Arrays de Objetos

Podemos usar forEach, map e outras funções para iterar por cada objeto do array.

```
turma.forEach(aluno => console.log(aluno.nome));
```

UNISAGRADO
Ensino Superior de Excelência

Extraindo Valores

Para pegar valores específicos, basta acessar o nome da propriedade.

```
let aluno = { nome: "Diego", idade: 20 };
console.log(aluno.nome); // "Diego"
```

UNISAGRADO
Ensino Superior de Excelência

Dicas e Resumo

Objetos ajudam a organizar dados com significado.

Arrays de objetos são úteis para listas, como turmas, produtos, etc.

Use métodos de arrays (map, forEach) para manipular dados de forma eficiente.

UNISAGRADO
Ensino Superior de Excelência

Módulo 6: Assincronia e APIs

Introdução à Assincronia

Assíncrono significa que o código não trava enquanto espera algo (como uma resposta do servidor).

Imagine pedir uma pizza: você não fica parado esperando, certo? Vai fazendo outras coisas.

Em JavaScript, isso evita travar a página, deixando-a mais fluida.

Trabalhando com APIs

API é como um “garçom” entre nosso app e uma fonte de dados.

Com uma API, você pode buscar dados externos (como clima, notícias) de um servidor.

Exemplo: Buscar clima usando uma API pública.

UNISAGRADO
Ensino Superior de Excelência

Entendendo o Fetch

fetch() é a função que chamamos para fazer uma requisição à API.

```
fetch("https://api.exemplo.com")
    .then(res ⇒ res.json())
    .then(data ⇒ console.log(data));
```

UNISAGRADO
Ensino Superior de Excelência

Promises e Como Lidar

Uma Promise é como uma promessa: "vou te dar o dado, mas quando eu tiver".
fetch retorna uma Promise; usamos .then para pegar o resultado quando chegar.
Se não der certo, cai no catch.

UNISAGRADO
Ensino Superior de Excelência

Tratamento de Erros com Try e Catch

try/catch ajuda a lidar com erros e evita que o código quebre.

```
try {
    let resposta = await fetch("https://api.exemplo.com");
    let dados = await resposta.json();
} catch (erro) {
    console.error("Erro:", erro);
}
```

UNISAGRADO
Ensino Superior de Excelência

Integração Completa

Exemplo de chamada de API com fetch, usando async/await:

```
async function buscarDados() {
    try {
        let resposta = await fetch("https://api.exemplo.com");
        let dados = await resposta.json();
        console.log(dados);
    } catch (erro) {
        console.log("Erro ao buscar dados:", erro);
    }
}
```

UNISAGRADO
Ensino Superior de Excelência

Dicas para Trabalhar com Assincronia

Sempre trate erros ao usar APIs.

Use async/await para deixar o código mais limpo.

Lembre: Assincronia deixa o app mais rápido e o usuário feliz!

UNISAGRADO
Ensino Superior de Excelência

Obrigado pela atenção!

/tulioalves7 /tuliobragaalves7 /tulioalvess7

1 ”Tudo tem o seu tempo determinado, e há tempo para todo o propósito debaixo do céu.

UNISAGRADO
Ensino Superior de Excelência