MÓDULO IV

Introdução à Programação I

A Educação é o primeiro passo para um futuro melhor...

MÓDULO IV

Introdução ao JavaScript: Variáveis, Tipos de Dados e Operadores

O JavaScript é uma linguagem de programação amplamente utilizada no desenvolvimento de páginas da web interativas e dinâmicas. Neste artigo, exploraremos os conceitos básicos do JavaScript, incluindo variáveis, tipos de dados e operadores.

O que é o JavaScript?

O JavaScript é uma linguagem de script leve, que permite aos desenvolvedores criar elementos interativos em páginas da web. Ele é frequentemente usado em conjunto com HTML e CSS para fornecer uma experiência mais envolvente aos usuários. Com o JavaScript, podemos criar formulários dinâmicos, efeitos visuais, animações e até mesmo manipular conteúdo em tempo real.

MÓDULO IV

Como programar em JavaScript

Aprender a programar em JavaScript é muito simples.
Basta você ter um computador para acessar console do seu navegador preferido.

Mas o que raios é um console?

O console do navegador é uma ferramenta de desenvolvedor integrada aos navegadores da web que permite aos desenvolvedores interagirem com páginas da web e executarem comandos em JavaScript diretamente no navegador. Ele fornece uma interface onde você pode visualizar mensagens de depuração, erros, avisos e resultados de códigos executados em tempo real.

A principal função do console do navegador é facilitar o processo de desenvolvimento e depuração de sites e aplicativos da web. Ele permite que os desenvolvedores inspecionem e analisem o código HTML, CSS e JavaScript em uma página, bem como façam testes e modifiquem o comportamento do código sem a necessidade de editar diretamente os arquivos fonte.

MÓDULO IV

Variáveis em JavaScript

As variáveis são usadas para armazenar dados que podem ser usados mais tarde em nosso código.
Em JavaScript, podemos declarar variáveis usando as palavras-chave *var*, *let* ou *const*. Por exemplo:

```
var idade = 30;
let nome = "Maria";
const PI = 3.14;
```

Aqui, *idade*, *nome* e *PI* são as variáveis, e atribuímos a elas um valor numérico, uma string e uma constante, respectivamente.
Vamos falar sobre a diferença entre os três tipos de variáveis do JavaScript futuramente.

MÓDULO IV

Abra uma nova janela, ou uma nova guia de seu navegador preferido (recomendamos o Google Chrome, Mozilla Firefox ou Microsoft Edge). Pressione a tecla *F12* ou clique com o botão direito do mouse na página e selecione *Inspecionar*.

Isso abrirá a ferramenta de desenvolvedor do navegador.

Procure a guia *Console(a)* na janela que se abriu.

No console, você verá um prompt onde pode digitar seu código JavaScript.

O símbolo > geralmente indica que você pode começar a digitar.

Vamos digitar no console a primeira variável que vimos acima:

```
var idade = 30;
```

MÓDULO IV

Tecle enter. Agora digite o nome da variável.

```
idade
```

O número 30 será printado no console.

```
> var idade = 30;
< undefined
> idade
< 30
```

Repita o exercício com *let nome* e *const PI*.

```
> var idade = 30;
< undefined
> idade
< 30
> let nome = "Maria";
< undefined
> nome
< 'Maria'
> const PI = 3.14;
< undefined
> PI
< 3.14
```

MÓDULO IV

Tipos de Dados em JavaScript

JavaScript é uma linguagem dinamicamente tipada, o que significa que o tipo de dado de uma variável é determinado automaticamente com base no valor atribuído a ela. Se você já programou em linguagens estaticamente tipadas, como Java ou C, sentirá certa diferença na declaração de variáveis.

Alguns dos tipos de dados em JavaScript incluem:

- Números: usado para representar valores numéricos, como 10, 3.14, etc.
- Strings: uma sequência de caracteres, como "Olá, mundo!".
- Booleanos: representando verdadeiro (true) ou falso (false).
- Arrays: uma coleção de elementos que podem ser de diferentes tipos.
- Objetos: representam entidades com propriedades e métodos.

MÓDULO IV

Operadores em JavaScript

Os operadores em JavaScript são usados para realizar ações em variáveis e valores. Alguns dos operadores mais comuns incluem:

- Operadores Aritméticos: como adição (+), subtração (-), multiplicação (*), divisão (/) e outros.
- Operadores de Atribuição: como "=" para atribuir valores a variáveis.
- Operadores de Comparação: como "==", "!=", "<", ">" para comparar valores.
- Operadores Lógicos: como "&&" (E), "||" (OU), "!" (NÃO) para realizar operações lógicas.

MÓDULO IV

Exemplo de Uso de Operadores

Como dizem, a prática leva a perfeição.
No console, crie as seguintes variáveis (lembre-se de teclar enter depois de criar cada variável):

```
let a = 5;
let b = 10;

let soma = a + b; // Resultado: 15
let subtracao = a - b; // Resultado: -5
let comparacao = a == b; // Resultado: false
let logica = (a < 0) && (b > 0); // Resultado: false
```

MÓDULO IV

```
> let a = 5;
< undefined
> let b = 10;
< undefined
> a + b
< 15
> a - b
< -5
> a / b
< 0.5
> a * b
< 50
> a == b
< false
```

Agora, em qualquer momento que você digitar uma das variáveis, o resultado será printado na tela.

MÓDULO IV

Em JavaScript podemos declarar variáveis de duas formas atualmente, com *let* ou *var*, sendo *let* a forma recomendada. *let* permite declarar variáveis que existem apenas no escopo no qual desejamos utilizá-las. Esse comportamento é diferente quando utilizamos *var*, que cria variáveis globais e que podem ser acessadas em qualquer local no arquivo ou função onde foram declaradas.

Exemplo 1

Variáveis podem armazenar diferentes tipos de valores como textos ou números:

```
1  let tipoNumber = 12
2  let tipoString1 = 'Texto entre aspas simples'
3  let tipoString2 = "Texto entre aspas duplas"
4  let tipoNull = null
5  let tipoUndefined = undefined
6  let tipoObject = { nome : 'José da Silva', idade : 18 }
7  let tipoBoolean = true || false
```

MÓDULO IV

Exemplo 2

É possível atribuir a uma variável o resultado de uma expressão:

```
1  let totalBruto = 5000
2  let desconto = 0.1
3
4  let totalLiquido = totalBruto - (totalBruto * desconto)
```

No código acima, uma vez que utilizamos parênteses, primeiro será calculada a multiplicação e, em seguida, a subtração. Após, a variável *totalLiquido* será iniciada com o resultado da operação matemática.

MÓDULO IV

Exemplo 3

As variáveis declaradas com *let* dentro de blocos *if* ou *for* não podem ser acessadas de fora destes blocos.

```
let admin = true

if(admin) {
    let menu = [ "home", "contato", "settings" ]
}

if(menu.includes("settings")) {
    // Alguma logica
}
```

Caso isso aconteça o script será encerrado com o erro abaixo:

```
if(menu.includes("settings")) {
^

ReferenceError: menu is not defined
```

MÓDULO IV

Exemplo 4

Quando usamos *var* para declarar a variável usada como índice em um *for* a mesma permanece acessível, mesmo após o encerramento deste laço de repetição.

```
1  for(var i = 0; i < precos.length; i++) {
2      total += precos[i]
3  }
4
5  media = total / precos.length
6
7  console.log(i) // 3
```

MÓDULO IV

Para corrigir esse código devemos utilizar o *let* no *for* para evitar que variáveis criadas para a sua execução sejam acessíveis fora de seu escopo:

```
1  for(let i = 0; i < precos.length; i++) {
2      total += precos[i]
3  }
4
5  media = total / precos.length
6
7  console.log(i) // ReferenceError: i is not defined
```

MÓDULO IV

Constantes

O valor de uma constante não pode ser modificado.

No JavaScript declaramos constantes com a palavra reservada *const*.

Uma constante se comporta em relação ao escopo onde foi declarada da mesma forma que uma variável declarada com *let*.

MÓDULO IV

Imagine uma caixa:

- **Variável (let):** É como uma caixa que você pode abrir e trocar o conteúdo quantas vezes quiser. Você pode colocar um brinquedo, depois um livro, depois roupas, etc.
- **Constante (const):** É como uma caixa que você só pode abrir uma vez para colocar um item dentro. Depois de fechada, você nunca mais pode trocar o que está lá dentro. O conteúdo da caixa fica fixo.

No mundo do JavaScript:

- Quando você usa let, você está criando uma variável que pode ter seu valor alterado ao longo do tempo.
  Por exemplo, você pode começar com let idade = 25; e depois mudar para idade = 26;.
- Quando você usa const, você está criando uma constante que recebe um valor fixo no momento da criação.
  Por exemplo, const PI = 3.14159;. Uma vez que você atribui esse valor a PI, você não pode mudá-lo. Se você tentar fazer PI = 3.15;, o JavaScript vai te dar um erro.

MÓDULO IV

Por que usar constantes?

- **Segurança:** Elas evitam que você mude acidentalmente um valor que deveria permanecer fixo, como o valor de PI ou configurações importantes do seu programa.
- **Clareza:** Elas deixam claro para quem lê o código que aquele valor não vai mudar, facilitando a compreensão do programa.

Em resumo:

- let = variável mutável (pode mudar).
- const = variável imutável (não pode mudar).

MÓDULO IV

Sintaxe

```
1 | const [ identificador ] = [ valor | expressão ]
```

- **identificador**: nome da constante
- **valor**: valor inicial que pode ser o resultado de uma expressão (opcional)

MÓDULO IV

```
1 | const [ identificador ] = [ valor | expressão ]
```

- const: Indica que estamos declarando uma constante, ou seja, um valor que não pode ser alterado após a atribuição inicial.
- [identificador]: Essa é a parte da *desestruturação*. Estamos criando uma constante chamada identificador que receberá um valor de um array. As colchetes [] indicam que estamos lidando com um array.
- [valor | expressão]: Essa é a parte que fornece o valor inicial para a constante. Ela também é um array. O valor pode ser um valor literal (como um número, string, etc.) ou o resultado de uma expressão (como uma operação matemática, chamada de função, etc.).

MÓDULO IV

Como funciona a desestruturação de *array*?

A *desestruturação* de *array* permite que você extraia valores de um *array* e os atribua a variáveis (ou constantes) de forma mais concisa. O identificador dentro das colchetes recebe o valor correspondente da mesma posição no *array* [valor | expressão].

Exemplo:

```
const numeros = [10, 20, 30];

const [primeiro] = numeros;

console.log(primeiro); // Saída: 10
```

Neste exemplo:

- numeros é um *array* com os valores 10, 20 e 30.
- const [primeiro] = numeros; desestrutura o *array* numeros. O valor na primeira posição (10) é atribuído à constante primeiro.

MÓDULO IV

Desestruturando múltiplos valores:

Você também pode desestruturar múltiplos valores de um *array*:

```
const coordenadas = [42.3601, -71.0589];
const [latitude, longitude] = coordenadas;
console.log(latitude); // Saída: 42.3601
console.log(longitude); // Saída: -71.0589
```

Ignorando valores:

Se você quiser ignorar algum valor do *array*, pode deixar a posição correspondente vazia:

```
const dados = [1, 2, 3, 4, 5];
const [primeiro, , terceiro, , quinto] = dados;
console.log(primeiro); // Saída: 1
console.log(terceiro); // Saída: 3
```

MÓDULO IV

Usando com expressões:

Você também pode usar expressões para gerar o array que será desestruturado:

```
const [resultado] = [2 + 3];

console.log(resultado); // Saída: 5
```

A desestruturação de array com const é uma forma poderosa e concisa de extrair valores de arrays e atribuí-los a constantes, tornando seu código mais legível e organizado.

MÓDULO IV

Exemplo 1

Podemos atribuir a uma constante o resultado de uma expressão:

```
const peso = 80.5

const altura = 1.80

const imc = peso / (altura *altura)

console.log(imc)
```

É comum usarmos constantes para evitar que um valor seja sobrescrito ao longo do código.

MÓDULO IV

Exemplo 2

Constantes são muito utilizadas para criar funções em conjunto da sintaxe das funções em flecha:

```
const desconto =
(preco, aliquota) =>
preco * aliquota
desconto(560.56, 0.08)
```

Criamos aqui uma função chamada desconto que retorna o cálculo do preço vezes a alíquota informada como parâmetro.

MÓDULO IV

Exemplo 3

Uma constante declarada sem ser iniciada com um valor gera um erro de sintaxe:

```
1 | const aliquota
2 |
3 | console.log(aliquota)
```

```
1 | (function (exports, require, module, __filename, __dirname) { const aliquota
2 |   ^^^^^^^^
3 |
4 | SyntaxError: Missing initializer in const declaration
```

MÓDULO IV

Exemplo 4

Assim como quando utilizamos let, uma constante está limitada ao escopo em que foi declarada:

Dentro do *if* a constante quantidade é tratada como uma nova declaração e o valor atribuído a ela poderá ser acessado apenas dentro desse bloco.

```
const quantidade = 10

if (quantidade >= 3) {

const quantidade = 20

console.log(quantidade) // 20

}

console.log(quantidade) // 10
```

MÓDULO IV

Exemplo 5

Um declarado como constante não pode ser sobrescrito:

```
const produto = { id : 1, nome : 'Grampo', preco : 34.7 }
produto = null
```

Sendo este o caso, o script será encerrado com um erro:

```
1  produto = null
2      ^
3  TypeError: Assignment to constant variable.
```

MÓDULO IV

Contudo, os atributos de um objeto declarado como constante podem ser sobrescritos.

```
1 const produto = { id : 1, nome : 'Grampo', preco : 34.7 }
2 produto.preco = null
```

Esse comportamento pode não ser desejado. Nesse caso podemos utilizar a função *Object.freeze()* para evitar que os atributos de um objeto sejam reescritos:

```
const produto = { id : 1, nome : 'Grampo', preco : 34.7 }

const produtoFreeze = Object.freeze(produto)

produtoFreeze.preco = null

console.log(produtoFreeze.preco) // 34.7
```

Embora o código acima não chegue a falhar o valor do atributo preco em *produtoFreeze* permanecerá o mesmo.

MÓDULO IV

Imagine...

Imagine um objeto como um "pacote" de informações:

- No nosso caso, o objeto produto é um pacote que contém informações sobre um produto:
- **id**: o número de identificação do produto (1).
- **nome**: o nome do produto ('Grampo').
- **preco**: o preço do produto (34.7).

O que o *Object.freeze()* faz?

- O *Object.freeze()* é como colocar um "lacre" nesse pacote. Depois de lacrado, você não pode mais:
- Adicionar novas informações ao pacote.
- Remover informações do pacote.
- Alterar os valores das informações que já estão no pacote.

MÓDULO IV

Nosso código:

- const produto = { id: 1, nome: 'Grampo', preco: 34.7 };: Criamos o objeto produto com as informações do produto.
- const produtoFreeze = Object.freeze(produto);: Usamos Object.freeze() para "lacrar" o objeto produto e armazenamos o resultado em produtoFreeze.
- produtoFreeze.preco = null;: Tentamos mudar o preço do produto para null. Como o objeto está lacrado, essa mudança não é permitida.
- console.log(produtoFreeze.preco);: Imprimimos o preço do produto no console. O valor impresso será 34.7, pois a tentativa de mudar o preço para null falhou.

MÓDULO IV

Em resumo:

- O Object.freeze() protege um objeto de qualquer alteração.
- Depois de usar Object.freeze(), você não pode adicionar, remover ou modificar as propriedades do objeto.
- No nosso exemplo, o preço do produto permanece 34.7 porque o objeto foi lacrado com Object.freeze().

Por que usar Object.freeze()?

- Segurança: Evita que você mude acidentalmente os valores de um objeto que deveria permanecer inalterado.
- Previsibilidade: Garante que o objeto sempre terá os mesmos valores, facilitando a depuração e a manutenção do código.

MÓDULO IV

undefined

Uma variável, constante ou função é undefined quando ela ainda não foi declarada.
Quando uma variável é utilizada sem que tenha sido declarada o script é encerrado com um erro:

```
1 | let desconto = 0.1
2 |
3 | preco = preco * desconto
```

Nesses casos o script será encerrado com um erro:

```
1 | preco = preco * desconto
2 | ^
3 | ReferenceError: preco is not defined
```

MÓDULO IV

Caso tenhamos declarado uma variável ou constante sem atribuir a ela um valor o script será encerrado com um erro:

```
1  var preco
2  let desconto
3
4  console.log(preco)      // undefined
5  console.log(desconto)   // undefined
```

MÓDULO IV

1. Exemplo com **const** (Constante)

Objetivo: Mostrar que *const* não pode ser reatribuída, mas pode ter propriedades modificadas (em objetos/arrays).

```
// ***** EXEMPLO 1: CONSTANTES (const) *****
// 'const' é usado para valores que NÃO devem ser reatribuídos (constantes)

// Exemplo 1.1: Constante primitiva (não pode mudar)
const PI = 3.14; // Declara uma constante PI
console.log("Valor de PI:", PI); // Mostra no console: 3.14

// PI = 3.1415; // DESCOMENTE ESTA LINHA PARA VER O ERRO! Constantes não podem ser

// Exemplo 1.2: Constante com objeto (pode mudar propriedades internas)
const pessoa = {
  nome: "João",
  idade: 15
};

console.log("Pessoa original:", pessoa);

pessoa.idade = 16; // ✅ PODE mudar propriedades de objetos/arrays
console.log("Pessoa modificada:", pessoa); // Idade atualizada para 16

// pessoa = { nome: "Maria" }; // ❌ ERRO! Não pode reatribuir a constante.
```

MÓDULO IV

2. Exemplo com ***let*** (Variável Mutável)

Objetivo: Mostrar que ***let*** pode ser reatribuída e tem escopo de bloco.

```
// ***** EXEMPLO 2: LET (Variável com escopo de bloco) *****
// 'let' é usado para valores que PODEM ser reatribuídos, mas só existem no bloco

// Exemplo 2.1: Reatribuição
let pontos = 10; // Declara 'pontos' com valor 10
console.log("Pontos iniciais:", pontos);

pontos = 20; // ✅ Pode ser reatribuída
console.log("Pontos atualizados:", pontos);

// Exemplo 2.2: Escopo de bloco
if (true) {
  let mensagem = "Só existo aqui dentro!"; // 'let' só existe neste bloco
  console.log("Dentro do bloco:", mensagem); // Funciona
}

// console.log(mensagem); // ❌ ERRO! 'mensagem' não existe fora do bloco.
```

MÓDULO IV

3. Exemplo com ***var*** (Variável Antiga)

Objetivo: Mostrar que ***var*** tem escopo global/função e pode causar confusão.

```
// ***** EXEMPLO 3: VAR (Variável antiga - EVITE USAR!) *****
// 'var' tem escopo de função ou global e pode causar bugs inesperados

// Exemplo 3.1: Escopo estranho
if (true) {
  var cor = "azul"; // 'var' vaza para fora do bloco!
  console.log("Dentro do bloco:", cor); // Mostra azul
}
console.log("Fora do bloco:", cor); // ❌ Funciona (mas não deveria!)

// Exemplo 3.2: Redeclaração (outro problema)
var idade = 15;
var idade = 20; // ✅ 'var' permite redeclarar (isso é ruim!)
console.log("Idade:", idade); // Mostra 20 (confuso!)
```

MÓDULO IV

4. Comparação Direta: const vs let vs var

Objetivo: Mostrar as diferenças lado a lado.

```
// ===== RESUMO: QUANDO USAR CADA UM =====

// Use CONST para valores FIXOS (constantes):
const ESCOLA = "Escola Estadual";
// ESCOLA = "Outra Escola"; // ❌ Erro!

// Use LET para valores que mudam DENTRO DE UM BLOCO:
let nota = 7;
nota = 8; // ✅ Permitido

// Evite VAR (ela age de forma inesperada):
var cidade = "São Paulo";
var cidade = "Rio"; // ✅ Permitido (mas pode causar bugs)
```

MÓDULO IV

1. **Demonstre no Console:**

- Peça para os alunos testarem cada exemplo no console do navegador (F12 > Console).

2. **Erros Propositais:**

- Mostre o que acontece ao tentar reatribuir const ou acessar let fora do bloco.

3. **Analogias:**

- const = Placa de trânsito (fixa).
- let = Lousa (pode apagar e reescrever).
- var = Giz em qualquer lugar (bagunça!).

Por que Evitar *var*?

❌ Vaza do bloco (*if/for*).
❌ Permite redeclaração.
❌ Dificulta a manutenção do código.

MÓDULO IV

Exercícios Práticos

1. Constante Imutável

Crie uma constante ANIMAL_FAVORITO com o valor "cachorro" e tente reatribuir para "gato". O que acontece?

```
// Seu código aqui
const ANIMAL_FAVORITO = "cachorro";
// ANIMAL_FAVORITO = "gato"; // Descomente para ver o erro!
```

2. Contador de Likes

Use let para criar um contador de likes que começa em 0 e aumenta em 1 a cada "curtida".

```
// Seu código aqui
let likes = 0;
likes += 1; // Primeira curtida
console.log(likes); // Deve mostrar 1
```

MÓDULO IV

3. Troca de Variáveis

Use let para trocar os valores entre timeA ("Flamengo") e timeB ("Vasco").

```
// Seu código aqui
let timeA = "Flamengo";
let timeB = "Vasco";
let temp = timeA;
timeA = timeB;
timeB = temp;
console.log(timeA, timeB); // Deve mostrar "Vasco Flamengo"
```

4. Escopo de Bloco

Use let dentro de um bloco if e tente acessar a variável fora dele. O que acontece?

```
if (true) {
  let materiaFavorita = "Matemática";
  console.log(materiaFavorita); // Funciona?
}
// console.log(materiaFavorita); // Descomente para ver o erro!
```

MÓDULO IV

5. Problema com var

Compare *var* e *let* em um loop. Qual é a diferença?

```
for (var i = 0; i < 3; i++) {
  setTimeout(() => console.log("var:", i), 100); // Mostra 3, 3, 3 (❌)
}
for (let j = 0; j < 3; j++) {
  setTimeout(() => console.log("let:", j), 100); // Mostra 0, 1, 2 (✅)
}
```

6. Lista de Compras

Use *const* com um *array* para adicionar itens a uma lista de compras.

```
const listaCompras = ["pipoca", "refri"];
listaCompras.push("chocolate"); // Isso funciona?
console.log(listaCompras); // Mostra ["pipoca", "refri", "chocolate"]
```

MÓDULO IV

7. Calculadora Simples

Use *const* para operações e *let* para o resultado.

```
const NUM1 = 10;
const NUM2 = 5;
let resultado = NUM1 + NUM2;
console.log(resultado); // Deve mostrar 15
```

8. Reatribuição com *let*

Crie uma variável nota com valor 6 e depois mude para 8.

```
let nota = 6;
nota = 8;
console.log("Nova nota:", nota); // Deve mostrar 8
```

MÓDULO IV

9. Const com Objeto

Modifique uma propriedade de um objeto declarado com const.

```
const celular = { modelo: "iPhone", ano: 2022 };
celular.ano = 2023; // Isso funciona?
console.log(celular); // Mostra { modelo: "iPhone", ano: 2023 }
```

10. Desafio: Jogo de Adivinhação

Combine const (número secreto) e let (tentativas) em um jogo.

```
const NUMERO_SECRETO = 7;
let tentativas = 3;
let palpite = parseInt(prompt("Adivinhe o número (1-10):"));

if (palpite === NUMERO_SECRETO) {
  alert("Acertou! 🎉");
} else {
  tentativas--;
  alert(`Errou! Tentativas restantes: ${tentativas}`);
}
```

MÓDULO IV

EXERCÍCIOS...

A Educação é o primeiro passo para um futuro melhor...

MÓDULO IV

FIM...

A Educação é o primeiro passo para um futuro melhor...