MÓDULO VI

Introdução à Programação I

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

MÓDULO VI

Operadores lógicos e de comparação

Para criar condições, usamos operadores de comparação e lógicos. Eles são as ferramentas que vão permitir que você verifique se algo é *maior*, *menor*, *igual*, ou *combina* várias condições ao mesmo tempo.

Operadores de comparação:

- **==:** Igual a
- **===:** Estritamente igual (compara valor e tipo)
- **!=:** Diferente de
- **!==:** Estritamente diferente (valor e tipo)
- **<, >, <=, >=:** Menor, maior, menor ou igual, maior ou igual

Operadores lógicos:

- **&&:** E (ambas as condições devem ser verdadeiras)
- **||:** Ou (pelo menos uma condição deve ser verdadeira)
- **!:** Não (inverte o resultado da condição)

MÓDULO VI

Estruturas condicionais no JavaScript
Como utilizar a estrutura condicional "*if* (se)" no JavaScript

Vamos aprender a utilizar as estruturas condicionais no JavaScript, começando pelo uso do **if(se), else if e else(senão).**

Primeiramente precisamos entender a sintaxe, podemos simplificar com o exemplo abaixo:

```
if (idade > 18){
    console.log("Você é obrigado a votar");
}
```

No exemplo nós utilizamos a palavra *if* seguido da condição *idade > 18* . Ou seja, caso a variável *idade* seja maior que 18 (verdadeiro), os comandos que estarão no bloco serão executados, se tratando do exemplo, será executado o comando *console.log("Você é obrigado a votar").*

MÓDULO VI

```
> let idade = 19;
<· undefined
> if(idade >18){
  console.log("Você é obrigado a votar");
  }
  Você é obrigado a votar
<· undefined
>
```

Porém este exemplo não está completo, afinal, há quem não pode votar e também o voto opcional. Podemos utilizar os operadores lógicos na condição para deixar mais robusto o nosso exemplo:

```
if (idade > 15 && idade < 18 || idade > 70){
    console.log("O voto é opcional");
}
```

MÓDULO VI

Agora nós criamos uma estrutura condicional utilizando *if* e os operadores lógicos *&&* (*E*) e *||*(*OU*), mas ainda podemos deixar mais completo o nosso exemplo utilizando a estrutura *else if* :

```
if (idade > 15 && idade < 18 || idade > 70) {
    console.log("O voto é opcional")
}else if (idade < 16) {
    console.log("Você não pode votar");
}
```

Como se pode ver, utilizamos o else if para impor outro bloco de comandos caso a primeira condição do if não fosse atendida, neste caso nós teremos o seguinte resultado:

MÓDULO VI

Contudo, para finalizar o nosso exemplo, precisamos também informar quando o voto é obrigatório. Percebe-se que é, justamente, o intervalo de *idade* que as condições que estão em nosso *if / else if* não estão abrangendo, portanto, podemos somente adicionar um *else*, ficando desta forma:

```
let idade = 17;
if (idade > 15 && idade < 18 || idade > 70) {
  console.log("O voto é opcional");
} else if (idade < 16) {
  console.log("Você não pode votar");
}
// Saída (com idade = 17):
// O voto é opcional

idade = 12;
if (idade > 15 && idade < 18 || idade > 70) {
  console.log("O voto é opcional");
} else if (idade < 16) {
  console.log("Você não pode votar");
}
// Saída (com idade = 12):
// Você não pode votar
```

```
if (idade > 15 && idade < 18 || idade > 70) {
    console.log("O voto é opcional")
}else if (idade < 16) {
    console.log("Você não pode votar");
}else {
    console.log("Você é obrigado a votar");
}
```

MÓDULO VI

Obtendo o seguinte resultado:

```
let idade = 25;
if (idade > 15 && idade < 18 || idade > 70) {
  console.log("O voto é opcional");
} else if (idade < 16) {
  console.log("Você não pode votar");
} else {
  console.log("Você é obrigado a votar");
}
// Saída:
// Você é obrigado a votar
```

Neste exemplo mostramos como é possível atribuir um comportamento de tomada de decisão utilizando as estruturas condicionais ***if, else if*** e ***else*** no *JavaScript*.

Em algumas situações onde trabalhamos com inúmeros ***if / else*** e com valores pré-definidos nas condições, podemos utilizar a instrução *Switch Case*.

MÓDULO VI

Estrutura if...else

Sintaxe

```
if (condição) {
  // código a ser executado se a condição for verdadeira
} else {
  // código a ser executado caso contrário
}
```

Exemplo prático:

Imagine que estamos desenvolvendo uma página para a plataforma da formação.

Queremos exibir uma mensagem motivadora ao aluno, dependendo do número de projetos que ele já completou.

```
> let projetosCompletos = 13;

  if (projetosCompletos === 13) {
    console.log("Parabéns, dev! Você completou todos os 13 projetos e está pronto para brilhar no mercado!");
  } else {
    console.log(`Falta pouco! Continue avançando e conquiste os ${13 - projetosCompletos} projetos restantes para completar seu portfólio.`);
  }
```

Aqui, verificamos se o aluno completou todos os 13 projetos da formação.
Se sim, ele recebe uma mensagem de parabéns. Caso contrário, ele é incentivado a continuar codando!

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

MÓDULO VI

Estrutura *else if*

Às vezes, você pode ter mais de duas condições para verificar.
Para isso, existe a estrutura *else if*, que permite adicionar condições extras à sequência de decisões.

Sintaxe

```
if (condição1) {
  // código a ser executado se a condição1 for verdadeira
} else if (condição2) {
  // código a ser executado se a condição2 for verdadeira
} else {
  // código a ser executado se nenhuma das condições for verdadeira
}
```

MÓDULO VI

Exemplo prático:

Agora, imagine que estamos ajustando o sistema de conquistas da formação.
Dependendo da quantidade de desafios resolvidos, o sistema pode exibir diferentes níveis de progresso.

```
> let desafiosResolvidos = 45;

  if (desafiosResolvidos >= 40) {
    console.log("Você é um mestre dos códigos! Já resolveu mais de 40 desafios!");
  } else if (desafiosResolvidos >= 20) {
    console.log("Ótimo progresso, dev! Você está no caminho certo, para resolver todos os desafios.");
  } else {
    console.log("Ainda há muitos desafios pela frente! Bora codar?");
  }
```

Nesse exemplo, o sistema ajusta a mensagem de acordo com o progresso do dev em resolver os desafios propostos. Assim, o aluno se sente incentivado a continuar evoluindo.

MÓDULO VI

Estrutura *switch*

Sintaxe

```
switch (expressão) {
  case valor1:
    // código a ser executado se a expressão for igual a valor1
    break;
  case valor2:
    // código a ser executado se a expressão for igual a valor2
    break;
  default:
    // código a ser executado se nenhum valor corresponder
}
```

MÓDULO VI

Exemplo prático:

Vamos brincar um pouco com as faixas salariais de programadores.
Imagine que você está desenvolvendo um sistema que, com base no nível de experiência do dev, mostra uma estimativa de salário.

```
let nivel = "pleno";

switch (nivel) {
  case "junior":
    console.log("Salário estimado para dev júnior: R$ 3.000 - R$ 4.000");
    break;
  case "pleno":
    console.log("Salário estimado para dev pleno: R$ 5.000 - R$ 8.000");
    break;
  case "senior":
    console.log("Salário estimado para dev sênior: R$ 9.000 - R$ 12.000");
    break;
  default:
    console.log("Nível não reconhecido. Bora estudar e subir de nível!");
}
// Saída:
// Salário estimado para dev pleno: R$ 5.000 - R$ 8.000
```

Dependendo do nível do dev (júnior, pleno ou sênior), o sistema exibe uma estimativa de salário. Se o nível informado não for reconhecido, incentivamos o dev *a continuar aprendendo.*

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

MÓDULO VI

Operador ternário

Sintaxe

```
condição ? valorSeVerdadeiro : valorSeFalso;
```

Exemplo prático:

Vamos usar o operador ternário para verificar se o aluno da Formação Full Stack já completou o curso ou ainda está em andamento.

```
> let formacaoCompleta = false;

  let mensagem = formacaoCompleta ? "Parabéns! Você completou a formação Full Stack e está pronto para brilhar
  no mercado!" : "Continue firme! Complete todos os projetos da formação Full Stack e conquiste seu espaço.";

  console.log(mensagem);
```

Nesse exemplo, se a formação estiver completa, o aluno recebe uma mensagem de parabéns. Caso contrário, é incentivado a continuar estudando e praticando.

MÓDULO VI

Como utilizar Switch Case no JavaScript

Outra estrutura condicional do JavaScript que podemos utilizar é o **Switch Case**. O *Switch Case* é uma instrução que se comporta de forma semelhante ao *if / else*, porém possui uma estrutura mais organizada e de fácil compreensão. *Lembrando que só é aceito valores pré-definidos e não expressões condicionais.*

Podemos analisar a estrutura do *Switch Case* no seguinte exemplo:

```
let signo = 'Leão';

switch (signo) {
  case 'Áries':
    console.log("De 21 março a 20 abril");
    break;
    case 'Touro':
    console.log("de 21 abril a 20 maio");
    break;
  case 'Gêmeos':
    console.log("de 21 maio a 20 junho");
    break;
  case 'Câncer':
    console.log("de 21 junho a 22 julho");
    break;
  case 'Leão':
    console.log("de 23 julho a 22 agosto");
    break;
  case 'Virgem':
    console.log("de 23 agosto a 22 setembro");
    break;
  default:
    console.log("Signo não registrado");
    break;
}
```

MÓDULO VI

Neste caso nós temos uma variável que recebe um signo, e então, devemos exibir qual o intervalo de nascimento referente a esse signo. Seria muito mais complexo utilizar *if / else* para este exemplo do que o *switch case*.

Ao analisar o exemplo, ao lado do *switch* atribuímos a variável signo, neste caso ele pede a **expressão** que será comparada a cada *case*. Sendo verdadeiro com alguma das opções, os comandos do bloco são executados até atingir a palavra-chave *break*. Caso nenhuma das opções seja verdadeira, a instrução irá executar os comandos que estarão na opção *default*.

Ao executar o exemplo acima, obtemos o seguinte resultado:

```
let signo = 'Leão';

switch (signo) {
  case 'Áries':
    console.log("De 21 de março a 20 de abril");
    break;
  case 'Touro':
    console.log("De 21 de abril a 20 de maio");
    break;
  case 'Gêmeos':
    console.log("De 21 de maio a 20 de junho");
    break;
  case 'Câncer':
    console.log("De 21 de junho a 22 de julho");
    break;
  case 'Leão':
    console.log("De 23 de julho a 22 de agosto");
    break;
  case 'Virgem':
    console.log("De 23 de agosto a 22 de setembro");
    break;
  default:
    console.log("Signo não registrado");
    break;
}
// Saída:
// De 23 de julho a 22 de agosto
```

MÓDULO VI

📜 O que o código faz?

É como um **"cardápio de signos"** que mostra as datas de cada um.
O programa:

1. Pega o signo ('Leão' no exemplo).
2. Procura a **opção certa** numa lista de casos.
3. Mostra o período correspondente.
4. Se não achar o signo, avisa que não está registrado.

🔍 Explicação passo a passo:

1. Variável signo:

```
let signo = 'Leão';  // Define o signo que queremos consultar
```

- É como escolher um filme no menu da Netflix 📺.

MÓDULO VI

2. Estrutura do switch:

```
switch (signo) {  // Verifica o valor da variável `signo`
```

- O switch é como um **detetive** que compara o signo com cada case.

3. Casos (case):

Cada case é uma **condição específica**:

```
case 'Leão':
    console.log("de 23 julho a 22 agosto");  // Resposta para Leão
    break;  // Sai do switch após achar a opção certa
```

- Funciona assim:
  - Se signo == 'Leão' → Mostra as datas do Leão.
  - O break é como um **"PARE, já encontramos!"** 🛑.

MÓDULO VI

4. default:

```
default:
    console.log("Signo não registrado");  // Se nenhum caso for atendido
```

- É o **plano B**: aparece se o signo não estiver na lista.

🌟 **Comparação com o mundo real:**

- **switch** → Como um **cardápio de pizza** 🍕:
  - Você pede "Calabresa" (case 'Calabresa'),
  - O chef vê qual é a receita certa,
  - Se não achar, avisa "Não temos" (default).

💡 **Por que usar switch?**

- **Mais organizado** que vários if/else quando há muitas opções.
- **Fácil de atualizar**: Basta adicionar novos cases.

MÓDULO VI

❓ Perguntas para fixar:

1. O que acontece se tirarmos um break?
   → **Resposta**: O código "vaza" e executa os casos abaixo até achar outro break!
2. Como adicionar o signo de "Libra"?

```
case 'Libra':
    console.log("de 23 setembro a 22 outubro");
    break;
```

🎯 Dica para ensino:

- Peça para **criarem um switch com seus próprios temas**:
  - Times de futebol 🏆,
  - Estilos de música 🎵,
  - Personagens favoritos 🎮.

MÓDULO VI

Estruturas de repetição no JavaScript

Agora vamos aprender a utilizar as estruturas de repetição no JavaScript, como: *while, for, for...in e for...of.*

Como utilizar **while**(enquanto) **no JavaScript**

Podemos utilizar a estrutura de repetição while caso seja necessário repetir um bloco de comandos por N vezes.

```
let i = 0;
while(i < 11){
    console.log('5 x ' + i + ' = ' + 5*i);
    i++;
}
```

MÓDULO VI

A sintaxe do *while*, como vemos no exemplo acima, é bem simples. Utilizamos a palavra reservada *while* , logo em seguida precisamos passar uma *condição* que será atendida enquanto ela retornar *true*, ou seja, *verdadeiro*. Neste caso, simulamos a tabuada do **5**, a variável ***i*** é o nosso contador, iniciamos ela com o valor ***0*** , portanto a *condição* será verdadeira até que ela atinja o valor ***11***, e por isso ela é iterada a cada repetição ***i++***, ao atingir tal valor o laço é quebrado.

Executando o código teremos o seguinte resultado:

```
let i = 0;
while (i < 11) {
  console.log("5 x " + i + " = " + 5 * i);
  i++;
}
// Saída:
// 5 x 0 = 0
// 5 x 1 = 5
// 5 x 2 = 10
// 5 x 3 = 15
// 5 x 4 = 20
// 5 x 5 = 25
// 5 x 6 = 30
// 5 x 7 = 35
// 5 x 8 = 40
// 5 x 9 = 45
// 5 x 10 = 50
```

MÓDULO VI

🎮 Explicação com Analogia de Videogame

Imagina que você é um robô repetindo uma tarefa até cumprir uma missão!

```
let i = 0;  // Você começa no nível 0 do jogo
while (i < 11) {  // Enquanto não chegar no nível 11...
    console.log('5 x ' + i + ' = ' + 5*i);  // Mostra a tabuada do 5 no nível atual
    i++;  // Avança para o próximo nível (i = i + 1)
}
```

🔍 Passo a Passo Detalhado

Início do Jogo (let i = 0)

Você começa com i = 0 (primeiro nível).

Condição do Loop (i < 11)

O robô pergunta: *"Eu já cheguei no nível 11?"*

Se **NÃO**, ele executa o bloco de código.

Corpo do Loop (console.log)

Mostra a mensagem com a tabuada do 5 (ex: 5 x 3 = 15).

Incremento (i++)

Você sobe um nível (i aumenta em 1).

Repete até...

Quando i = 11, o robô para porque a condição i < 11 vira falsa.

MÓDULO VI

💡 Por que isso é útil?

- **Automatiza tarefas repetitivas** (como calcular todas as tabuadas de uma vez).
- **Base para jogos**: Criar inimigos que aparecem em waves, contagem regressiva, etc.

🎯 Exercício Prático

1. Fazer a tabuada do 7.
2. Parar no número 15 (em vez de 10).

```
let i = 0;
while (i < 16) {  // Altere o limite aqui
    console.log('7 x ' + i + ' = ' + 7*i);  // Mude o 5 para 7
    i++;
}
```

MÓDULO VI

🚀 Dica de Ensino

- **Mostre no Console**: Peça para alguém digitar o código no console do navegador (F12 > Console) e ver a mágica acontecer!
- **Relacione com Tik Tok**: *"É como um vídeo que se repete até você rolar para o próximo."*

📊 Tabela de Valores (Para Visualizar)

Nível (`i`)	Cálculo (`5*i`)	Saída no Console
0	5 x 0 = 0	`5 x 0 = 0`
1	5 x 1 = 5	`5 x 1 = 5`
...	...	...
10	5 x 10 = 50	`5 x 10 = 50`

❓ Perguntas para Fixação

1. O que acontece se esquecer *i++*?
   → **Resposta**: Loop infinito! O robô fica preso no mesmo nível.
2. Como mostrar só resultados pares?
   → **Dica**: Use *if (i % 2 === 0)* dentro do loop.

MÓDULO VI

Utilizando Do-While (faça - enquanto)

O *do-while* tem o mesmo conceito que o *while* com uma única diferença, a condição é verificada após os comandos do bloco serem executados, ou seja, mesmo que a condição seja falsa, é garantia que o bloco será executado ao menos uma vez.

Podemos testar da seguinte forma:

```
let contador = 0;

do{
    console.log("O contador vale: " + contador);
    contador++;
}while(contador == 1)
```

MÓDULO VI

A variável contador foi inicializada com o valor *0* e o *while* está com a condição *contador == 1* ou seja, a condição é *falsa* e utilizando o while nosso bloco de comandos não seria executado. Porém podemos perceber que ao utilizar *do-while*, o bloco vem antes da verificação da condição, com isso ele garante que o bloco de comandos seja executado ao menos uma vez.

Desta forma vamos obter o seguinte comportamento:

```
> let contador = 0;

  do{
      console.log("O contador vale: " + contador);
      contador++;
  }while(contador == 1)
  O contador vale: 0
  O contador vale: 1
< 1
```

MÓDULO VI

🌟 **Explicação Passo a Passo:**

1. **Inicialização**:

```
let contador = 0;  // Começamos com contador em 0
```

2. **do { ... }:**

- **Executa o bloco de código pelo menos UMA VEZ**, mesmo se a condição for falsa.
- É como dizer: *"Faça isso primeiro, depois verifique se repete!"*.

3. **Corpo do Loop**:

```
console.log("O contador vale: " + contador);  // Mostra o valor atual
contador++;  // Aumenta o contador em 1 (0 → 1 → 2)
```

MÓDULO VI

4. Condição (while):

```
} while (contador < 2);  // Repete enquanto o contador for MENOR que 2
```

📜 O que acontece no código?

Passo	Contador	Saída no Console	Condição (contador < 2)
1	0	"O contador vale: 0"	0 < 2 → **Verdadeiro**
2	1	"O contador vale: 1"	1 < 2 → **Verdadeiro**
3	2	*(Não executa)*	2 < 2 → **Falso**

MÓDULO VI

💡 Por que usar do-while?

- **Garante que o código rode pelo menos 1 vez** (útil para menus, jogos, etc.).
- Exemplo real:
  "Tente digitar a senha até acertar, mas pelo menos peça UMA vez!" 🔐

🎮 Analogia com Videogame:

- É como um **chefão que te ataca PELO MENOS UMA VEZ**, mesmo que você já esteja com a vida zerada 💀 → Depois, ele só repete se você não derrotá-lo.

❓ Perguntas para Fixar:

1. **Quantas vezes o console.log é executado?**
   → **Resposta**: 2 vezes (para contador = 0 e contador = 1).
2. **O que acontece se mudarmos para while (contador < 0)?**
   → **Resposta**: O bloco roda **1 vez** mesmo assim (mostra "O contador vale: 0")!

MÓDULO VI

🔄 Comparação com while normal:

do-while	while
Executa **primeiro**, verifica depois	Verifica **antes** de executar
Útil para menus interativos	Útil para contagens controladas

🛠️ Teste Prático:

Mude a condição para contador < *5* e veja quantas vezes repete:

```
let contador = 0;
do {
    console.log("Contador agora é: " + contador);
    contador++;
} while (contador < 5);  // Experimente outros números!
```

MÓDULO VI

Como utilizar *For* (para) no JavaScript

A estrutura de repetição *for* no JavaScript segue o mesmo princípio que o *while*, porém este recurso é mais utilizado quando se sabe o número de iterações da repetição, como ao percorrer um vetor, por exemplo.

Podemos replicar o exemplo da tabuada, porém utilizando *for*:

```
for(let i = 0; i < 11; i++){
    console.log("5 x " + i + " = " + 5*i);
}
```

Para utilizar o *for*, usamos a palavra-chave *for* seguida de três declarações, primeiramente iniciamos a variável que será o controlador do nosso laço de repetição, logo em seguida precisamos informar a condição a ser atendida e, por último, uma expressão que será executado ao final de cada iteração do *for*, normalmente utilizamos para incrementar a variável que será utilizada como controlador do nosso laço.

MÓDULO VI

Ao executar o código teremos o resultado semelhante ao exemplo utilizando *while*:

```
for (let i = 0; i < 11; i++) {
  console.log("5 x " + i + " = " + 5 * i);
}
// Saída:
// 5 x 0 = 0
// 5 x 1 = 5
// 5 x 2 = 10
// 5 x 3 = 15
// 5 x 4 = 20
// 5 x 5 = 25
// 5 x 6 = 30
// 5 x 7 = 35
// 5 x 8 = 40
// 5 x 9 = 45
// 5 x 10 = 50
```

OBS: Vale ressaltar que o *while* é mais indicado para situações onde não temos conhecimento do número de iterações do laço, já o *for* é mais adequado para situações que o número de iterações está definido previamente.

MÓDULO VI

🌟 Explicação Passo a Passo:

1. **Inicialização (let i = 0)**
   - É como começar um jogo no **nível 0**.
   - i é nosso contador que vai de 0 até 10.
2. **Condição (i < 11)**
   - O jogo continua **enquanto** o nível for menor que 11.
   - Tradução: *"Repita até o nível 10!"*.
3. **Atualização (i++)**
   - No final de cada rodada, **sobe um nível** (i aumenta em 1).
   - i++ é igual a i = i + 1.
4. **Corpo do Loop (console.log)**
   - Mostra a tabuada do 5 para o nível atual.
   - Exemplo: Se i = 3 → Mostra "5 x 3 = 15".

MÓDULO VI

📜 O que acontece em cada etapa?

Nível (`i`)	Cálculo (`5*i`)	Saída no Console	Condição (`i < 11`)
0	5 x 0 = 0	`5 x 0 = 0`	0 < 11 → ✅
1	5 x 1 = 5	`5 x 1 = 5`	1 < 11 → ✅
...	...	...	...
10	5 x 10 = 50	`5 x 10 = 50`	10 < 11 → ✅
11	-	*(Não executa)*	11 < 11 → ❌

💡 Por que usar *for*?

- **Economiza linhas**: Tudo (início, condição e incremento) fica numa linha só.
- **Controle total**: Você sabe exatamente quantas vezes o loop vai rodar.

MÓDULO VI

🎮 Analogia com TikTok:

É como um **vídeo que se repete automaticamente**:

1. **Início**: Você clica no vídeo (i = 0).
2. **Condição**: Enquanto não chegar no último vídeo (i < 11).
3. **Ação**: Assiste o vídeo atual (console.log).
4. **Atualização**: Passa para o próximo vídeo (i++).

🔄 Comparação com while

for	while
Melhor para **contagens exatas**	Melhor para **condições complexas**
Tudo (início, condição, incremento) numa linha	Início e incremento ficam separados

MÓDULO VI

🛠️ Teste Você Mesmo!

Mude o código para:

1. **Tabuada do 7**:

```
for (let i = 0; i < 11; i++) {
  console.log("7 x " + i + " = " + 7*i);
}
```

2. **Contar de 10 em 10**:

```
for (let i = 0; i <= 100; i += 10) {
  console.log(i);
}
```

❓ Perguntas para Fixar:

1. **Quantas vezes o console.log executa?**
   → **Resposta**: 11 vezes (de 0 até 10).
2. **O que acontece se tirarmos i++?**
   → **Resposta**: Loop infinito! O contador nunca aumenta e fica preso no i = 0.

MÓDULO VI

🎯 **Dica para Ensino:**

Peça para alguém criar um **"gerador de tabuadas"**:

Ele cria uma **tabuada interativa** onde o usuário digita um número e o programa mostra a tabuada completa desse número de 0 a 10.

```
const numero = parseInt(prompt("Digite um número para a tabuada:"));
for (let i = 0; i <= 10; i++) {
  console.log(`${numero} x ${i} = ${numero*i}`);
}
```


```
const numero = parseInt(prompt("Digite um número para a tabuada:"));
for (let i = 0; i <= 10; i++) {
  console.log(`${numero} x ${i} = ${numero * i}`);
}
```


Vamos explicar esse exemplo...

MÓDULO VI

🌟 Explicação Detalhada:

1. Pedindo o Número ao Usuário:

```
const numero = parseInt(prompt("Digite um número para a tabuada:"));
```

- **prompt**: Abre uma caixinha no navegador para o usuário digitar um número (ex: "7").
- **parseInt**: Converte o texto digitado em número (ex: "7" → 7).
- **const numero**: Armazena o número digitado para usar na tabuada.

2. Loop for (Repetição):

```
for (let i = 0; i <= 10; i++) {
```

- **let i = 0**: Começa a contagem em 0.
- **i <= 10**: Repete enquanto i for menor ou igual a 10 (vai até 10).
- **i++**: Aumenta i em 1 a cada volta (0, 1, 2, ..., 10).

MÓDULO VI

3. Mostrando a Tabuada:

```
console.log(`${numero} x ${i} = ${numero * i}`);
```

- **Template String (crases)** Permite misturar texto com variáveis.
- **${numero}**: Mostra o número digitado pelo usuário.
- **${i}**: Mostra o contador atual (0 a 10).
- **${numero * i}**: Calcula e mostra o resultado (ex: 7 x 2 = 14).

💡 Por que usar esse código?

- **Interativo**: O usuário escolhe qualquer número para a tabuada.
- **Eficiente**: O *for* automatiza os cálculos repetitivos.
- **Didático**: Ótimo para aprender loops e templates strings.

MÓDULO VI

🎮 Analogia com TikTok:

É como um **vídeo que mostra todos os "likes" de 1 a 10** para um post:

1. Você pergunta:
   "Quantos likes quer ver?" (prompt).
2. O sistema mostra:
   Post x 1 like = 1 like
   Post x 2 likes = 2 likes
   ... até 10.

❓ Perguntas para Fixar:

1. **O que acontece se o usuário digitar "abc"?**
   → **Resposta**: NaN (Not a Number), porque não é um número válido.
2. **Como mostrar só resultados pares?**
   → **Modifique o loop**:

```
for (let i = 0; i <= 10; i += 2) { ... }
```

MÓDULO VI

Utilizando *for...in* (/para em) e *for...of (para de)*

No JavaScript, nós temos mais duas variações da utilização do *for*, o *for...in* e *for...of*.

O *for...in* é utilizado para percorrer as propriedades de um objeto, por exemplo:

```
let carro = {
marca: 'ford',
modelo: 'fiesta',
cor: 'prata',
}

for (propriedade in carro) {
    console.log(propriedade);
}
```

Criamos uma instância do objeto carro, onde temos como propriedade a marca, modelo e cor.

Ao utilizar o for, passamos a variável que irá percorrer as propriedades do objeto e logo em seguida, de qual objeto

MÓDULO VI

O resultado esperado:

```
let carro = {
  marca: 'ford',
  modelo: 'fiesta',
  cor: 'prata'
};

for (let propriedade in carro) {
  console.log(propriedade);
}
// Saída:
// marca
// modelo
// cor
```

Também podemos acessar os atributos de cada propriedade, para isso basta usarmos propriedade como índice do nosso objeto, desta maneira:

```
for (propriedade in carro) {
    console.log(carro[propriedade]);
}
```

MÓDULO VI

Vamos obter a seguinte saída:

```
let carro = {
  marca: 'ford',
  modelo: 'fiesta',
  cor: 'prata'
};

for (let propriedade in carro) {
  console.log(propriedade);
}
// Saída:
// marca
// modelo
// cor

for (let propriedade in carro) {
  console.log(carro[propriedade]);
}
// Saída:
// ford
// fiesta
// prata
```

MÓDULO VI

E por fim, podemos exibir o objeto por completo somente usando o *for...in*:

```
for (propriedade in carro) {
    console.log(propriedade + ': ' + carro[propriedade]);
}
```

```
let carro = {
  marca: 'ford',
  modelo: 'fiesta',
  cor: 'prata'
};

for (let propriedade in carro) {
  console.log(propriedade);
}
// Saída:
// marca
// modelo
// cor

for (let propriedade in carro) {
  console.log(carro[propriedade]);
}
// Saída:
// ford
// fiesta
// prata

for (let propriedade in carro) {
  console.log(propriedade + ': ' + carro[propriedade]);
}
// Saída:
// marca: ford
// modelo: fiesta
// cor: prata
```

MÓDULO VI

O *for...of* nós podemos utilizar para percorrer objetos iteráveis como *Maps*, *Sets* e *Vetores* de forma simples e eficaz, da seguinte forma:

```
let carros = [ 'fiesta', 'onix', 'fusca', 'saveiro' ];

for (let carro of carros) {
    console.log(carro);
}
```

Neste caso, vamos obter os valores de cada índice:

```
> let carros = [ 'fiesta', 'onix', 'fusca', 'saveiro' ];
< undefined
> for (let carro of carros) {
      console.log(carro);
  }
  fiesta
  onix
  fusca
  saveiro
```

MÓDULO VI

🔍 Código Explicado:

```
let carros = ['fiesta', 'onix', 'fusca', 'saveiro']; // 1 Lista de carros
for (let carro of carros) { // 2 Loop mágico
  console.log(carro); // 3 Mostra cada carro
}
```

🌟 Passo a Passo:

1. **Criamos a Lista** (*let carros = [...]*)
   - É como fazer uma **playlist** com 4 carros:
     ['fiesta', 'onix', 'fusca', 'saveiro']
2. **Loop *for...of***
   - Funciona como **passar as músicas uma a uma**:
     - *let* carro: Cria uma variável temporária (como "música atual").
     - of carros: Pega cada item da lista carros.

MÓDULO VI

3. **Mostra no Console** (console.log)
   - Imprime cada carro da lista, um por vez:

```
fiesta
onix
fusca
saveiro
```

💡 **Por que usar *for...of*?**

- **Simples**: Não precisa contar índices como no *for* tradicional.
- **Legível**: Parece frase em português: *"Para cada **carro de carros**, faça..."*.
- **Perfeito para listas**: *Arrays*, *strings*, etc.

MÓDULO VI

Também é possível exibir o índice referente a cada item usando o método *entries()* da seguinte forma:

```
for (let [index, carro] of carros.entries()) {
    console.log(index, carro);
}
```

Desta maneira vamos obter os índices e os respectivos valores:

```
> for (let [index, carro] of carros.entries()) {
      console.log(index, carro);
  }
  0 "fiesta"
  1 "onix"
  2 "fusca"
  3 "saveiro"
< undefined
```

MÓDULO VI

🎮 Analogia com TikTok:

É como **rolar o feed**:

- Cada carro é um **vídeo** que aparece na sua tela.
- O loop *for...of* é seu **dedo deslizando** para ver o próximo.

🔄 Diferença para for comum:

for...of (Este exemplo)	for tradicional
Acessa **os valores** diretamente	Precisa usar índices (carros[i])
Mais simples para listas	Mais controle (ex: pular itens)

MÓDULO VI

🛠️ **Teste Você Mesmo!**

Modifique o código para:

Listar frutas:

```
let frutas = ['maçã', 'banana', 'uva'];
for (let fruta of frutas) {
  console.log(fruta);
}
```

Mostrar letras de uma palavra:

```
let palavra = "JavaScript";
for (let letra of palavra) {
  console.log(letra);
}
```

MÓDULO VI

❓ Perguntas para Fixar:

1. O que acontece se a lista estiver vazia?
   → Resposta: O loop não executa (nada é mostrado).
2. Como adicionar " - *Novo*" em cada carro?
   → Modifique o *console.log*:

```
console.log(carro + " - Novo");
```

🎯 Dica para Ensino:

Peça para alguém criar uma **lista de games** e usar *for...of* para mostrar cada um:

```
let games = ['Minecraft', 'Fortnite', 'Roblox'];
for (let game of games) {
  console.log(`Jogo: ${game}`);
}
```

MÓDULO VI

EXERCÍCIOS...

MÓDULO VI

FIM...

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