Aula 4

O que vamos aprender nessa aula:

- Quebra de fluxo e tomada de decisão
- Condicionais IF/ELSE
- Condicional ternário (? :)
- Operadores Lógicos (&& ||)
- Conceito de Truthy/Falsy
- Exercicios IF/ELSE e Operadores Lógicos
- Condicionais SWITCH
- Operadores de Coalescência (?? e ?.)

Quebra de fluxo e tomada de decisão

A quebra de fluxo e a tomada de decisão são conceitos fundamentais em JavaScript.
Na programação, muitas vezes é necessário que o código execute diferentes ações com base em condições específicas.

A quebra de fluxo refere-se à capacidade de interromper a execução normal do código e desviar para um caminho diferente, dependendo de certas condições. Isso permite que o programa tome decisões com base em informações ou eventos específicos.

A tomada de decisão é realizada usando estruturas condicionais, como o `if` / `else` e o `switch`. Com essas estruturas, é possível avaliar uma condição e executar diferentes blocos de código com base no resultado dessa avaliação. Isso permite que o programa selecione o caminho apropriado de acordo com as circunstâncias.

Esses conceitos são essenciais para criar lógica complexa e interativa em JavaScript, permitindo que os desenvolvedores criem programas adaptáveis e responsivos.

Estrutura do IF

A estrutura condicional `if` é utilizada para avaliar uma condição e executar um bloco de código se essa condição for verdadeira.

```
if (condicao) {
    // código a ser executado se a condição for verdadeira
}
```

Dentro do bloco de código do `if`, podemos realizar qualquer ação desejada, como atribuir valores a variáveis, exibir mensagens, chamar funções, entre outras. É importante garantir que a condição do `if` seja corretamente avaliada, pois somente se a condição for verdadeira é que o bloco de código será executado.

Essa estrutura é muito útil para executar ações específicas apenas quando determinada condição é atendida, permitindo a criação de lógica personalizada e adaptável em nossos programas.

Estrutura IF/ELSE

A estrutura condicional `if/else` é utilizada para avaliar uma condição e executar um bloco de código se essa condição for verdadeira. Caso a condição seja falsa, é possível definir um bloco de código alternativo para ser executado utilizando o `else`.

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

Com o `if/else`, podemos criar diferentes caminhos de execução no nosso programa, dependendo do resultado da condição. Se a condição do `if` for verdadeira, o bloco de código dentro do `if` será executado. Caso contrário, o bloco de código dentro do `else` será executado.

Essa estrutura nos permite tomar decisões com base em condições específicas, executando diferentes ações dependendo do resultado da avaliação da condição. É uma forma eficiente de criar lógica complexa e adaptável em nossos programas.

Estrutura ELSE IF

A estrutura condicional `else if` é utilizada para avaliar uma série de condições e executar diferentes blocos de código com base no resultado dessas avaliações. Essa estrutura é uma extensão do `if/else`, permitindo a criação de caminhos adicionais de execução.

```
if (condicao1) {
    // código a ser executado se a condicao1 for verdadeira
} else if (condicao2) {
    // código a ser executado se a condicao1 for falsa e a condicao2 for verdadeira
} else {
    // código a ser executado se todas as condições anteriores forem falsas
}
```

Com o `else if`, podemos adicionar múltiplas condições para serem avaliadas sequencialmente. Assim, o programa irá verificar cada condição em ordem e executar o bloco de código associado à primeira condição verdadeira encontrada. Caso nenhuma das condições seja verdadeira, o bloco de código dentro do `else` será executado.

Essa estrutura nos permite criar lógica mais complexa e granular, onde diferentes ações serão tomadas com base em cada condição avaliada. É uma forma eficaz de lidar com casos específicos e criar programas mais adaptáveis e flexíveis.

Operador Ternário (? :)

O operador ternário em JavaScript é uma forma concisa de realizar uma avaliação condicional em uma única linha. Ele é composto por três partes: a condição a ser avaliada, o valor a ser retornado se a condição for verdadeira e o valor a ser retornado se a condição for falsa.

A estrutura do operador ternário é a seguinte:

```
condicao ? valorSeVerdadeiro : valorSeFalso
```

Se a condição for verdadeira, o valorSeVerdadeiro será retornado. Caso contrário, o valorSeFalso será retornado.

O operador ternário é útil quando precisamos tomar uma decisão com base em uma condição simples, sem a necessidade de criar um bloco de código maior com o `if`/`else`. Ele torna o código mais conciso e legível em situações em que a lógica é simples e direta.

Exemplo de uso do operador ternário:

```
const idade = 18;
const podeDirigir = idade >= 18 ? 'Pode dirigir' : 'Não pode dirigir';
console.log(podeDirigir);
```

Neste exemplo, a condição `idade >= 18` é avaliada. Se for verdadeira, o valor 'Pode dirigir' será atribuído à variável `podeDirigir`. Caso contrário, o valor 'Não pode dirigir' será atribuído.

O operador ternário é uma ferramenta útil para simplificar a lógica condicional em situações simples, permitindo que você tome decisões e atribua valores de forma mais concisa e direta.

Operadores Lógicos (&& e ||)

Os operadores lógicos são utilizados para realizar operações de lógica booleana em JavaScript. Eles permitem combinar condições e avaliar múltiplas expressões para determinar o resultado final.

Operador Lógico E (&&)

O operador lógico E (`&&`) retorna `true` se todas as expressões fornecidas forem verdadeiras. Caso contrário, retorna `false`. Ele avalia as expressões da esquerda para a direita e interrompe a avaliação assim que uma expressão falsa é encontrada, pois, nesse caso, o resultado final já é garantido.

Exemplo de uso do operador lógico E:

```
const idade = 25;
const possuiCNH = true;

if (idade >= 18 && possuiCNH) {
    console.log('Pode dirigir');
} else {
    console.log('Não pode dirigir');
}
```

Neste exemplo, o operador lógico E é utilizado para verificar se a pessoa tem idade igual ou superior a 18 anos (`idade >= 18`) e se possui CNH (`possuiCNH`). Se ambas as condições forem verdadeiras, a mensagem `'Pode dirigir'` será exibida. Caso contrário, a mensagem `'Não pode dirigir'` será exibida.

Operador Lógico OU (||)

O operador lógico OU (`||`) retorna `true` se pelo menos uma das expressões fornecidas for verdadeira. Caso contrário, retorna `false`. Ele avalia as expressões da esquerda para a direita e interrompe a avaliação assim que uma expressão verdadeira é encontrada, pois, nesse caso, o resultado final já é garantido.

Exemplo de uso do operador lógico OU:

```
const diaSemana = 'sábado';
const diaUtil = diaSemana === 'segunda' || diaSemana === 'terça' || diaSemana === 'qua
rta' || diaSemana === 'quinta' || diaSemana === 'sexta';

if (diaUtil) {
    console.log('Dia útil');
} else {
    console.log('Fim de semana');
}
```

Neste exemplo, o operador lógico OU é utilizado para verificar se `diaSemana` é igual a algum dos dias úteis da semana. Se a condição for verdadeira, ou seja, se `diaSemana` for igual a 'segunda', 'terça', 'quarta', 'quinta' ou 'sexta', a mensagem `'Dia útil'` será exibida. Caso contrário, a mensagem `'Fim de semana'` será exibida.

Os operadores lógicos (`&&` e `||`) são ferramentas poderosas para combinar condições e criar lógica condicional complexa em JavaScript. Eles permitem que você tome decisões com base na avaliação de múltiplas expressões, tornando seu código mais expressivo e flexível.

Tabela verdade

Expressão 1	Expressão 2	Resultado (E)	Resultado (OU)
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Conceito de Truthy/Falsy

Em JavaScript, os conceitos de "truthy" e "falsy" são usados para descrever os valores que são considerados verdadeiros ou falsos em um contexto booleano. Quando uma expressão é avaliada em um contexto booleano, ela retornará `true` se for "truthy" e `false` se for "falsy".

Existem alguns valores específicos que são considerados "falsy" em JavaScript:

- `false`: o valor booleano falso.
- `0`: o número zero.
- `''` (string vazia): uma string sem caracteres.
- `null`: representa a ausência intencional de um valor.

- `undefined`: representa uma variável não atribuída ou uma propriedade não existente.
- `NaN`: representa um valor inválido em operações numéricas.

Todos os outros valores em JavaScript são considerados "truthy", o que significa que eles serão avaliados como `true` em um contexto booleano.

Esse conceito é importante ao lidar com estruturas condicionais, como o `if` e o `else`. Por exemplo, podemos usar o conceito de "truthy" e "falsy" para verificar se uma variável possui um valor válido antes de executar determinado bloco de código.

```
let nome = '';

if (nome) {
    console.log('O nome é válido');
} else {
    console.log('O nome é inválido');
}
```

Neste exemplo, a expressão `nome` é avaliada em um contexto booleano. Se `nome` for uma string não vazia (ou seja, "truthy"), a mensagem `'O nome é válido'` será exibida. Caso contrário, a mensagem `'O nome é inválido'` será exibida.

Os conceitos de "truthy" e "falsy" são úteis para criar lógica condicional mais flexível em JavaScript, permitindo que o programa tome decisões com base em valores que são considerados verdadeiros ou falsos em um contexto booleano.

Exercícios de `if-else`

1. **Exercício Fácil:**
   Escreva um programa em JavaScript que verifica se um número é positivo, negativo ou igual a zero. Use a estrutura `if-else` para realizar essa verificação.

2. **Exercício Moderado**:
   Escreva um programa em JavaScript que verifica se um número é par ou ímpar. Use a estrutura `if-else` para realizar essa verificação.

3. **Exercício Intermediário:**
   Escreva um programa em JavaScript que verifica se um número é divisível por 3 e por 5 ao mesmo tempo. Use a estrutura `if-else` para realizar essa verificação.

4. **Exercício Avançado:**
   Escreva um programa em JavaScript que verifica se um ano é bissexto. Use a estrutura `if-else` para realizar essa verificação.
   Dica: um ano bissexto é divisível por 4, mas não por 100, a menos que também seja divisível por 400.

5. **Exercício Desafiador:**
   Escreva um programa em JavaScript que verifica se um número é um *palíndromo*. Use a estrutura `if-else` para realizar essa verificação. Um *palíndromo* é um número que permanece o mesmo quando seus dígitos são invertidos.
   Dica: para inverter um numero utilize:
   `parseInt(numero.toString().split('').reverse().join(''))`

Estrutura SWITCH

A estrutura condicional `switch` é utilizada para avaliar uma expressão e executar diferentes blocos de código com base no valor dessa expressão. Essa estrutura é especialmente útil quando há múltiplas condições a serem verificadas e diferentes ações a serem tomadas para cada valor.

```
switch (expressao) {
    case valor1:
        // código a ser executado se a expressao for igual a valor1
        break;
    case valor2:
        // código a ser executado se a expressao for igual a valor2
        break;
    case valor3:
    case valor4:
        // código a ser executado se a expressao for igual a valor3 ou valor4
        break;
    default:
        // código a ser executado se nenhum dos casos anteriores for verdadeiro
```

```
        break;
}
```

No `switch`, a expressão é avaliada e comparada com cada caso. Se o valor da expressão coincidir com um caso específico, o bloco de código associado a esse caso será executado. O comando `break` é utilizado para sair do `switch` após a execução do bloco de código correspondente.

Podemos agrupar casos em conjunto, como no exemplo acima, onde `valor3` e `valor4` compartilham o mesmo bloco de código. Isso permite que executemos a mesma ação para múltiplos valores.

Caso a expressão não corresponda a nenhum dos casos, o bloco de código dentro do `default` será executado. O `default` é opcional e serve como uma ação padrão a ser executada caso nenhum caso anterior seja verdadeiro.

O `switch` é uma estrutura poderosa para lidar com diferentes cenários e tomar ações com base nos valores da expressão. Ele oferece uma forma mais concisa e organizada de lidar com múltiplas condições em comparação com a sequência de `if` / `else if` / `else`.

Operadores de Coalescência (??)

Os operadores de coalescência são utilizados para fornecer um valor padrão quando um valor é nulo ou indefinido. Existem dois tipos de operadores de coalescência em JavaScript: o operador de coalescência nula (`??`) e o operador de coalescência opcional (`?.`).

O operador de coalescência nula (`??`) retorna o valor à esquerda se não for nulo ou indefinido, caso contrário, retorna o valor à direita. Ele é útil para fornecer um valor padrão quando o valor original é nulo ou indefinido.

Exemplo de uso do operador de coalescência nula:

```
const nome = nomeUsuario ?? 'Usuário Desconhecido';
console.log(nome);
```

Neste exemplo, se `nomeUsuario` for nulo ou indefinido, o valor padrão `'Usuário Desconhecido'` será atribuído à variável `nome`.

Exemplo de uso dos operadores de coalescência

```
const valor1 = null;
const valor2 = undefined;
const valor3 = 'Valor existente';

const resultado1 = valor1 ?? 'Valor padrão';
const resultado2 = valor2 ?? 'Valor padrão';
const resultado3 = valor3 ?? 'Valor padrão';

console.log(resultado1);
console.log(resultado2);
console.log(resultado3);
```

Neste exemplo, o operador de coalescência nula é utilizado para fornecer um valor padrão quando `valor1` e `valor2` são nulos ou indefinidos. No caso de `valor3`, como ele possui um valor existente, o operador de coalescência nula não tem efeito e o valor original é retornado.

Os operadores de coalescência são úteis para garantir que tenhamos um valor válido em situações em que um valor pode ser nulo ou indefinido.

Operador de Coalescência Opcional (?.)

O operador de coalescência opcional (`?.`) é utilizado para acessar propriedades de um objeto de forma segura, evitando erros caso o objeto seja nulo ou indefinido. Ele permite encadear múltiplas propriedades de forma simplificada, verificando a existência de cada uma delas antes de acessá-las.

Exemplo de uso do operador de coalescência opcional:

```
const usuario = {
    nome: 'João',
    endereco: {
        rua: 'Rua A',
        cidade: 'São Paulo'
    }
};

const cidadeUsuario = usuario?.endereco?.cidade;
console.log(cidadeUsuario);

const estadoUsuario = usuario?.endereco?.estado;
console.log(estadoUsuario);
```

Neste exemplo, o operador de coalescência opcional é utilizado para acessar a propriedade `cidade` dentro do objeto `endereco` do objeto `usuario`. Se alguma das propriedades ao longo do caminho (`endereco` ou `cidade`) não existir, o valor retornado será `undefined`.

O operador de coalescência opcional simplifica o acesso seguro a propriedades em objetos, evitando erros e facilitando o tratamento de casos em que propriedades podem estar ausentes.

```
const nome = undefined;
const nomeCompleto = nome?.nomeCompleto ?? 'Nome não encontrado';
console.log(nomeCompleto);
```

Neste exemplo, a propriedade `nomeCompleto` é acessada dentro do objeto `nome`, que é `undefined`. Utilizamos o operador de coalescência opcional (`?.`) para realizar o acesso seguro à propriedade. Como `nome` é `undefined`, o valor padrão `'Nome não encontrado'` é atribuído à constante `nomeCompleto`.

Exercícios de `switch/case`

1. **Exercício Fácil:**
   Escreva um programa em JavaScript que recebe uma cor em inglês e retorna o seu nome em português. Utilize a estrutura `switch/case` para realizar essa verificação. As cores podem ser: "red" (vermelho), "blue" (azul), "green" (verde), "yellow" (amarelo) e "black" (preto).
2. **Exercício Moderado:**
   Escreva um programa em JavaScript que recebe um número de 1 a 12 e retorna o mês correspondente. Caso seja um número inválido retorna "Número inválido.” Utilize a estrutura `switch/case` para realizar essa verificação.
3. **Exercício Intermediário:**
   Escreva um programa em JavaScript que recebe uma letra do alfabeto e retorna se é uma vogal ou uma consoante. Utilize a estrutura `switch/case` para realizar essa verificação.
4. **Exercício Avançado:**
   Crie um programa em JavaScript que implemente uma calculadora básica. A função `calculadora` receberá dois números como parâmetro e um terceiro parâmetro indicando a ação a ser executada. Utilize a estrutura `switch/case` para

realizar a operação correta com base na ação fornecida. As ações possíveis são: "somar", "subtrair", "dividir" e "multiplicar". O resultado da operação deve ser retornado pela função.
Exemplo de uso: `calculadora(5, 3, "somar")` deve retornar 8.

5. **Exercício Desafiado**
   Escreva um programa em JavaScript que recebe um número de 1 a 7 e retorna o dia da semana correspondente. Utilize a estrutura `switch/case` para realizar essa verificação.
   Além disso, implemente uma verificação adicional utilizando `if-else` para retornar "Fim de semana" caso o número seja 1 ou 7 (domingo ou sábado), e "Dia útil" caso contrário.

   Exemplo de uso: `verificarDiaSemana(3)` deve retornar "Quarta-feira, Dia útil".

   Dica: Utilize os valores de 1 a 7 para representar os dias da semana, sendo 1 para domingo, 2 para segunda-feira e assim por diante.