Aula 5

O que vamos aprender nessa aula:

- Operador de Coalescência Nula
- Operador de Coalescência Opcional
- While Loop
- Do While Loop

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`.

❗ Problema

Precisamos criar um jogo da adivinhação, em que o usuário precisa acertar um número aleatório entre 0 e 100.

- O usuário terá um local para digitar a resposta.
- Se o usuário errar a resposta o programa deve responder se o numero é maior ou menos ao número informado pelo usuário.
- Quando o usuário acertar o programa vai retornar o número de tentativas.

exemplo:

```
Adivinhe o número entre 1 e 100:
> 50
O número é maior que 50.
> 75
O número é menor que 75.
> 65
O número é menor que 65.
> 60
O número é maior que 60.
> 62
Parabéns! Você acertou o número em 5 tentativas
```

While Loop

O loop "while" em JavaScript é usado para executar um bloco de código repetidamente enquanto uma condição especificada for verdadeira. A estrutura básica do loop "while" é a seguinte:

```
while (condição) {
    // código a ser executado
}
```

Aqui está um exemplo de um loop "while" que imprime os números de 1 a 5:

```
let i = 1;

while (i <= 5) {
    console.log(i);
    i++;
}
```

While com `break`

Neste exemplo, a condição é `i <= 5`. Enquanto essa condição for verdadeira, o bloco de código dentro do loop será executado. A variável `i` é incrementada em cada iteração para evitar um loop infinito.

```
let i = 1;

while (true) {
    console.log(i);

    if (i === 5) {
        break;
    }

    i++;
}
```

Neste exemplo, usamos um loop `while` que imprime os números de 1 a 5. A condição do loop é `true`, o que significa que o loop é executado indefinidamente. No entanto, usamos a declaração `if` para verificar se `i` é igual a 5. Se essa condição for verdadeira, usamos a palavra-chave `break` para sair do loop e interromper a execução.

Do-While Loop

O loop "do-while" em JavaScript é semelhante ao loop "while", mas com uma diferença importante. No loop "do-while", o bloco de código é executado pelo menos uma vez, antes de verificar a condição de continuação. A estrutura básica do loop "do-while" é a seguinte:

```
do {
    // código a ser executado
} while (condição);
```

Aqui está um exemplo de um loop "do-while" que imprime os números de 1 a 5:

```
let i = 1;

do {
    console.log(i);
    i++;
} while (i <= 5);
```

Neste exemplo, o bloco de código dentro do loop é executado uma vez, independentemente da condição. Em seguida, a condição é verificada. Se a condição for verdadeira, o loop continuará executando, caso contrário, o loop será encerrado.

Lembre-se de usar loops com cuidado para evitar loops infinitos e garantir que a condição de saída seja alcançada.

Exercícios de `while`

1. **Exercício Fácil:**
   Escreva um programa em JavaScript que imprima **todos os números** de 1 a 10 utilizando um loop `while`.
2. **Exercício Moderado:**
   Escreva um programa em JavaScript que calcule a soma dos números de 1 a 100 utilizando um loop `while`. Imprima o resultado no final.
3. **Exercício Intermediário:**
   Escreva um programa em JavaScript que encontre e imprima todos os **números primos** de 1 a 100 utilizando um loop `while`.

4. **Exercício Avançado:**
   Escreva um programa em JavaScript que calcule e i**mprima os primeiros N termos da sequência de Fibonacci,** onde N é um número fornecido pelo usuário, utilizando um loop `while`.
5. **Exercicios Desafiador:**

   Escreva um programa em JavaScript cria uma lista de contatos, com dados inputados pelo usuário, dados para salvar:

   - Nome
   - Telefone
   - Idade
   - Profissão

   O usuário deve inputar dados de 5 contatos.

   Ao final o programa deve mostrar uma lista com os contatos.