[CT JS] Aula 2

♣ Conteúdos

Vetores e Matrizes

Array: Estrutura de dados que permite armazenar vários valores em uma única variável

Principais caracteristicas:

- Indices: Utilizados para acessar os elementos da array.
- Comprimento (Length): indica o número de elementos da array.
- Métodos: funções acopladas as arrays para manipulação dos elementos
- Dinamismo: O tamanho do array pode ser alterado durante a execução

Métodos:

- **push**(): add um novo elemento no final do array e retorna o novo tamanho do array
- **unshift()** add um novo elemento no inicio do array e retorna o novo tamanho do array
- **indexOf()**: retorna a primeira posição do elemento buscado, se não encontrar retorna `-1`
- **lastIndexOf()**: retorna a última posição do elemento , se não encontrar retorna `-1`
- **includes()**: retorna se o array possui ou não o elemento (true or false)
- **find()**: retorna a primeira ocorrencia do elemento buscado `(array.find( a ⇒ a == 2))`;
- **filter()**: retorna um novo array com todos elementos da condição, não altera o array de origem.
- **pop()**: remove o ultimo item do array e retorna o item removido. Este método altera o array original.
- **shift()**: remove o primeiro item do array e altera o array de origem.

- **splice()**: modifica o conteúdo da array, adicionando ou removendo elementos
  - Inserir elementos:

```
let meuArray = [1, 2, 3, 4, 5]; meuArray.splice(2, 0, 6, 7); // Inserir os elementos 6 e 7 a partir do índice 2 console.log(meuArray); // Saída: [1, 2, 6, 7, 3, 4, 5]
```

  - Remover elementos:

```
let meuArray = [1, 2, 3, 4, 5]; meuArray.splice(2, 2); // Remover 2 elementos a partir do índice 2 console.log(meuArray); // Saída: [1, 2, 5]
```

  - Substituir elemento:

```
let meuArray = [1, 2, 3, 4, 5]; meuArray.splice(2, 2, 6, 7); // Remover 2 elementos a partir do índice 2 e inserir os elementos 6 e 7 console.log(meuArray); // Saída: [1, 2, 6, 7, 5]
```

- **fill()**: utilizado para preencher um array com um valor padrão.
- **concat()**: Concatena arrays e não modifica os originais. `const arr3 = arr1.concat(arr2);`
- **reverse()**: Inverte a ordem do array e ALTERA o array original
- **sort()**: Ordena o array original. Padrão crescente.

```
const numeros = [5, 2, 8, 1, 7]; //ordenação padrão numeros.sort(); // Função de comparação para ordenar em ordem decrescente const compararDecrescente = (a, b) => b - a; // Usando o sort com a função de comparação numeros.sort(compararDecrescente); console.log(numeros); // Saída: [8, 7, 5, 2, 1]
```

Matrizes: Array de arrays ou array bidimensional

Spread Operator

Dicionários e Objetos

Condicionais if/else

O `if` e o `else` são estruturas de controle de fluxo em programação que permitem que o código tome decisões com base em condições.

1. **If (Se):**
   - O `if` é usado para verificar se uma condição é verdadeira.
   - Se a condição for verdadeira, o bloco de código dentro do `if` é executado.
   - Se a condição for falsa, o bloco de código dentro do `if` é simplesmente ignorado.

   Exemplo:

```
let idade = 18; if (idade >= 18) { console.log("Você é maior de idade."); }
```

2. **Else (Senão):**
   - O `else` é usado em conjunto com o `if` para fornecer uma alternativa quando a condição do `if` é falsa.
   - Se a condição do `if` for falsa, o bloco de código dentro do `else` é executado.

   Exemplo:

```
let idade = 16; if (idade >= 18) { console.log("Você é maior de idade."); } else { console.log("Você é menor de idade."); }
```

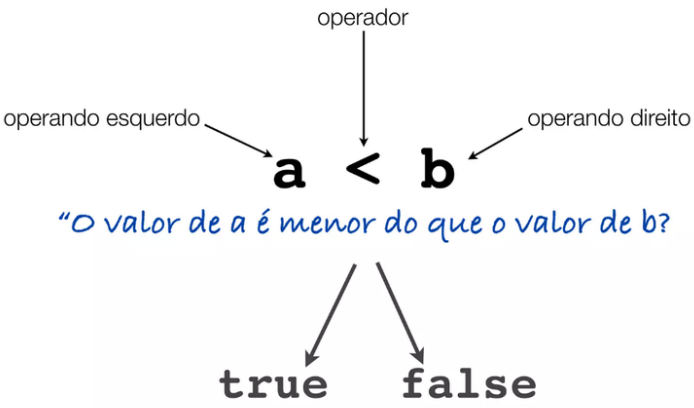
3. **If/Else (Se/Senão):**
   - O `if` e o `else` juntos permitem criar uma estrutura de decisão mais completa.
   - Se a condição do `if` for verdadeira, o bloco de código dentro do `if` é executado. Se for falsa, o bloco de código dentro do `else` é executado.

   Exemplo:

```
let temperatura = 25; if (temperatura >= 30) { console.log
("Está muito quente lá fora!"); } else { console.log("A temp
eratura está agradável."); }
```

Operadores

- **Comparação**



Operadores de comparação

Operador	Significado
==	igual
<	menor
>	maior
<=	menor ou igual
>=	maior ou igual
!=	diferente (*não igual*)

Operadores	Precedência
<, >, <=, >=	mais elevada
==, !=	menos elevada

- **Operadores Lógicos**: Tem menos precedência do que operadores de comparação

Operador	Significado
&&	AND (E)
||	OR (OU)
!	NOT (NEGAÇÃO)

Tabelas de verdade: AND

a	b	a && b
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
FALSE	FALSE	FALSE

Tabelas de verdade: OR

a	b	a || b
TRUE	TRUE	TRUE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	FALSE	FALSE

Tabelas de verdade: NOT

Condicionais ternários

As condicionais ternárias, também conhecidas como operadores ternários, são uma maneira concisa de escrever expressões condicionais em uma única linha. Elas são frequentemente usadas para atribuir valores com base em uma condição.

A estrutura básica de um operador ternário é:

```
condicao ? valorSeVerdadeiro : valorSeFalso;
```

- Se a condição for verdadeira, o operador ternário retorna `valorSeVerdadeiro`.
- Se a condição for falsa, o operador ternário retorna `valorSeFalso`.

Exemplo Simples:

```
let idade = 20; let status = (idade >= 18) ? "Maior de idade" :
"Menor de idade"; console.log(status);
```

A principal vantagem dos operadores ternários é a sua concisão, tornando o código mais curto e legível quando há apenas uma condição simples a ser avaliada.

Condicionais switch

O `switch` é uma estrutura de controle de fluxo em JavaScript que permite tomar decisões com base no valor de uma expressão. É uma alternativa ao `if-else` quando se deseja comparar uma única expressão com vários valores possíveis.

A estrutura básica do `switch` é a seguinte:

```
switch (expressao) { case valor1: // Código a ser executado se e
xpressao for igual a valor1 break; case valor2: // Código a ser
executado se expressao for igual a valor2 break; // Adicione mai
s cases conforme necessário default: // Código a ser executado s
e nenhum case corresponder a expressao }
```

Explicação Passo a Passo:

1. **Avaliação da Expressão:**
   - A expressão é avaliada uma vez.
   - O valor resultante é comparado com os valores em cada `case`.
2. **Comparação de Valores:**
   - Cada `case` contém um valor possível que a expressão pode ter.
   - Se o valor da expressão for igual a um `case`, o código dentro desse `case` é executado.
3. **Break:**
   - O `break` é usado para sair do `switch` após o código de um `case` ser executado.
   - Isso impede a execução dos códigos dos `case` seguintes.
4. **Default (Opcional):**
   - O `default` é opcional e é executado se nenhum dos `case` corresponder à expressão.
   - Funciona como um "caso contrário" para situações não previstas.

Exemplo:

```
let diaSemana = "quarta"; switch (diaSemana) { case "segunda": c
onsole.log("Dia de começar a semana!"); break; case "quarta": co
nsole.log("Metade da semana!"); break; case "sexta": console.log
("Quase lá, é sexta-feira!"); break; default: console.log("Outro
dia da semana."); }
```

Operadores de coalescencia

O operador de coalescência, também conhecido como "nullish coalescing" em inglês, é representado por `??` em JavaScript. Ele oferece uma maneira simples e conveniente de fornecer um valor padrão quando o valor à esquerda é `null` ou `undefined`.

A estrutura básica do operador de coalescência é:

```
valorÀEsquerda ?? valorPadrão;
```

- Se `valorÀEsquerda` não for `null` nem `undefined`, o operador retorna esse valor.
- Se `valorÀEsquerda` for `null` ou `undefined`, o operador retorna `valorPadrão`.

Exemplo Simples:

```
let nomeUsuario = null; let nomePadrao = "Visitante"; let nomeFi
nal = nomeUsuario ?? nomePadrao; console.log(nomeFinal);
```

Neste exemplo, se `nomeUsuario` for `null` ou `undefined`, `nomeFinal` receberá o valor padrão "Visitante". Se `nomeUsuario` tiver um valor diferente de `null` ou `undefined`, `nomeFinal` receberá esse valor.

O operador de coalescência é útil quando você deseja fornecer um valor padrão específico apenas se a variável estiver explicitamente `null` ou `undefined`, sem tratar outros valores falsy como `0`, `false` ou uma string vazia (`''`). Ele oferece uma alternativa mais específica e previsível ao operador lógico OR (`||`).

O operador `&&` é um operador lógico que avalia duas expressões e retorna o valor da última expressão se ambas forem verdadeiras. Ele é frequentemente usado para curto-circuitar avaliações quando o segundo operando só é avaliado se o primeiro for verdadeiro.

Em resumo, enquanto ambos `??` e `&&` podem ser utilizados em certas situações para controlar o fluxo baseado em condições, o `??` é especificamente projetado para coalescência nula, enquanto o `&&` é um operador lógico que retorna o último valor verdadeiro em uma cadeia de operações.

♧Exercícios

> Link para alunos
> [CT JS] Exercícios - Aula 2

▼ Arrays e Matrizes

1. **Desmontando o Castelo de cartas**

   Você está construindo um castelo de cartas e, de repente, percebe que a torre está muito alta e instável. Para evitar um desastre, decide remover a última carta da torre. Retorne um array com a torre modificada

```
const torreDeCartas = ["A", "2", "3", "4", "5", "J"];
```

```
//Resolução torreDeCartas.pop();
```

2. **Encontrando o Tesouro Perdido**

   Você está em uma expedição em busca de um tesouro perdido. Para localizar o tesouro, você tem um mapa com pistas. Retorne a posição do elemento que representa o tesouro.

```
const mapa = ["Floresta", "Montanha", "Caverna", "Tesouro",
"Rio"]; const tesouro = "Tesouro";
```

```
//Resolução mapa.indexOf(tesouro);
```

3. **Iniciando um jogo de Dominó**

   Você está jogando dominó e precisa remover a primeira peça da pilha de peças para dar início ao jogo.

```
const pilhaDomino = [ [3, 6], [1, 4], [5, 5], [2, 3] ];
```

```
//Resolução pilhaDomino.shift();
```

4. **Organizando Livros na estante**

   Você está organizando sua estante de livros. Retorne um array com os títulos de livros organizados em ordem alfabética.

```
const livros = ["Harry Potter", "Senhor dos Anéis", "1984",
"A Revolução dos Bichos"];
```

```
//Resolução const livros = ["Harry Potter", "Senhor dos Anéi
s", "1984", "A Revolução dos Bichos"]; livros.sort() consol
e.log(livros);
```

5. **Removendo itens indesejados**

   Você recebeu um array de números, mas quer remover todos os números pares. Retorna um novo array apenas com os números ímpares.

```
const numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

```
//Resolução const numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const novoArray = numeros.filter(numero => numero % 2 !==
0); console.log(novoArray);
```

▼ Operadores lógicos

Qual o resultado das seguintes comparações:

```
const a = 5; const b = 10; const c = 2; a > b && b < c; a < b &&
b < c; a < b || b < c; a > b && b > c; !(a > c); a < b || b < c
&& a < c;
```