Estruturas de Controle em JavaScript - Resumo e Explicação

Resumo curto para agenda

Aprendi a usar if/else, switch, loops (for, while, do/while), break/continue, e como iterar arrays e objetos para controlar o fluxo do programa.

Códigos Comentados e Explicações

1. Função simples com if/else

```
const imprimirResultado = function (nota) {
    if (nota >= 7) {
        console.log('Aprovado!'); // Mostra quando a nota é maior ou igual a 7
    } else {
        console.log('Reprovado!'); // Caso contrário
    }
}

imprimirResultado(10); // Aprovado!
imprimirResultado(4);  // Reprovado!
imprimirResultado('Epa!'); // Cuidado! Tipo não numérico pode gerar
comportamento inesperado
```

Explicação: if/else é usado para decisões simples, avaliando condições booleanas.

2. Loop `for...in` em arrays e objetos

```
const notas = [6.7, 7.4, 9.8, 8.1, 5.5, 3.0, 2.3, 10.0];
for(let i in notas) {
    console.log(`Nota = ${notas[i]}`); // Itera pelos índices do array
}

const pessoa = { nome: 'Ana', sobrenome: 'Silva', idade: 29, peso: 64 };
for(let atributo in pessoa) {
    console.log(`${atributo} = ${pessoa[atributo]}`); // Itera pelos atributos
do objeto
}
```

Explicação: `for...in` percorre chaves de objetos ou índices de arrays.

3. Loops tradicionais

```
let contador = 1;
while (contador <= 10) {
    console.log(`Contador = ${contador}`);
    contador++;
}

for(let i = 1; i <= 10; i++) {
    console.log(`i = ${i}`);
}
```

Explicação: `while` e `for` executam blocos repetidamente até que a condição não seja mais verdadeira.

4. `do...while`

```
function getInteirorAleatorioEntre(min, max) {
    const valor = Math.random() * (max - min) + min;
    return Math.floor(valor);
}
let opcao = 0;
do {
    opcao = getInteirorAleatorioEntre(-1, 10);
    console.log(`Opção escolhida foi ${opcao}.`);
} while (opcao != -1);
```

Explicação: `do...while` garante que o bloco execute ao menos uma vez antes de verificar a condição.

5. Break e continue

```
const nums = [1,2,3,4,5,6,7,8,9,10];
for(x in nums) {
    if(x == 5) break; // Sai do loop
    console.log(`${x} = ${nums[x]}`);
}
for(y in nums) {
    if(y == 5) continue; // Pula a iteração
    console.log(`${y} = ${nums[y]}`);
}
```

Explicação: `break` interrompe o loop, `continue` pula para a próxima iteração.

6. Switch case

```
const imprimirResultado = function (nota) {
    switch (Math.floor(nota)) {
        case 10:
        case 9:
            console.log('Quadro de Honra!');
            break;
        case 8:
        case 7:
            console.log('Aprovado!');
            break;
        case 6:
        case 5:
            console.log('Recuperação!');
            break;
        case 4:
        case 3:
        case 2:
        case 1:
        case 0:
            console.log('Reprovado!');
            break;
        default:
            console.log('Nota inválida!');
    }
}
```

Explicação: `switch` avalia uma expressão e executa o bloco do `case` correspondente. Permite agrupar múltiplos casos.

7. Loop com array e controle de fluxo

```
const notas = [6.7, 7.4, 9.8, 8.1, 7.7];
for(let i = 0; i < notas.length; i++) {
    console.log(`Nota = ${notas[i]}`);
}
```

Explicação: Loop `for` clássico percorrendo índices de array.

Resumo Final: Estes códigos mostram como controlar o fluxo do programa usando if/else, switch, loops, break/continue, e iterar arrays e objetos de forma eficiente.