JavaScript: Fundamentos para Desenvolvimento Web Interativo - Turma 2025B

3.4 Escopo

O escopo de funções refere-se ao contexto no qual variáveis e declarações de funções são acessíveis. É um conceito fundamental para entender como funções interagem com variáveis e com outros blocos de código. Vamos explorar os conceitos-chave do escopo de funções e como eles afetam o comportamento do seu código.

Escopo de Função

Em JavaScript, o escopo de função é o ambiente em que variáveis, parâmetros e declarações de função são acessíveis dentro de uma função. Tudo que é declarado dentro de uma função, como variáveis e funções internas, é local para essa função e não pode ser acessado de fora dela.

Veja um exemplo no quadro abaixo:

```
function minhaFuncao() {
    const x = 10; // Variável local ao escopo da função
    console.log(x);
}

minhaFuncao(); // Imprime 10
console.log(x); // Causa erro, pois `x` está no escopo da função
```

Neste exemplo, a variável **x** foi declarada dentro da função **minhaFuncao**. Portanto, ela só é acessível dentro dessa função. Tentar acessá-la fora do escopo da função resulta em erro.

Escopo global

O escopo global refere-se ao ambiente onde variáveis e funções são acessíveis em todo o programa. Qualquer variável ou função declarada fora de uma função pertence ao escopo global.

Veja um exemplo no quadro abaixo:

```
const y = 20; // Variável global

function outraFuncao() {
    console.log(y); // Acessa a variável global
}

outraFuncao(); // Imprime 20
```

Neste exemplo, a variável **y** é declarada no escopo global e é acessível dentro da função **outraFuncao**. As funções têm acesso ao escopo global, mas não vice-versa.

Escopo de bloco

Desde o ECMAScript 2015 (ES6), variáveis declaradas com **let** ou **const** têm escopo de bloco, significando que elas só são acessíveis no bloco onde foram declaradas. Os escopos de bloco são criados por estruturas como **if**, **for**, e **while**.

Veja um exemplo no quadro abaixo:

```
function bloco() {
    if (true) {
        let x = 10; // Escopo de bloco
        console.log(x); // Funciona
    }

    // console.log(x); // Causa erro, pois `x` está fora do escopo do bloco
}
```

Hoisting

Em JavaScript, as declarações de função e variáveis com var sofrem "hoisting". Isso significa que elas são "elevadas" para o início do escopo da função, independentemente de onde são declaradas no código. No entanto, apenas as declarações são elevadas, não as atribuições.

Veja um exemplo no quadro abaixo:

```
function exemplo() {
    console.log(a); // undefined, por causa do hoisting da variável `var`
    var a = 10; // Declaração da variável
    console.log(a); // Imprime 10
}
```

Escopo léxico

O escopo léxico refere-se ao contexto em que uma função é definida, determinando quais variáveis e funções estão acessíveis para ela. Funções podem "lembrar" do escopo em que foram definidas, mesmo quando são chamadas fora desse contexto. Este comportamento é também conhecido como "closures".

Veja um exemplo no quadro abaixo:

```
function funcaoExterna() {
    const valor = "externo";

    function funcaoInterna() {
        console.log(valor); // Acessa a variável do escopo da função externa
    }

    return funcaoInterna; // Retorna uma função que "lembra" do escopo

}

const minhaFuncao = funcaoExterna();
minhaFuncao(); // Imprime "externo", porque lembra do escopo léxico
```

Neste exemplo, a função **funcaoInterna** tem acesso ao escopo da função **funcaoExterna**, mesmo após **funcaoExterna** ser chamada e retornada. Isso é possível por causa do escopo léxico, que mantém referências ao contexto original onde a função foi definida.

◄ 3.3 Passagem de parâmetros

Seguir para...

3.5 Teste seus conhecimentos ►