JavaScript: Fundamentos para Desenvolvimento Web Interativo - Turma 2025B

3.3 Passagem de parâmetros

Ao definir uma função, você pode especificar uma lista de parâmetros entre parênteses. Esses parâmetros atuam como variáveis locais dentro da função, recebendo os valores passados quando a função é chamada.

Veja um exemplo no quadro abaixo:

```
function saudar(nome) {
    console.log("Olá, " + nome + "!");
}
```

Neste exemplo, **nome** é um parâmetro que a função **saudar** espera receber quando é chamada.

Passagem de parâmetrps por valor e por referência

Quando você passa parâmetros para uma função em JavaScript, é importante entender como o valor é transmitido para a função e como ele é tratado lá dentro. A forma como os parâmetros são passados pode afetar como eles são manipulados e se alterações feitas no valor dentro da função terão impacto fora dela.

Em JavaScript, a passagem de parâmetros para funções é por **valor** para tipos primitivos (como números, strings e booleanos) e por **referência** para objetos e arrays.

Passagem por valor

Quando você passa um parâmetro de tipo primitivo para uma função em JavaScript, como um número, uma string ou um booleano, a passagem é por valor. Isso significa que a função recebe uma cópia do valor do argumento. Assim, qualquer modificação que você faça no parâmetro dentro da função não afetará o valor original fora dela. Veja no quadro abaixo abaixo um exemplo para ilustrar essa situação:

```
function adicionarDez(x) {
    x = x + 10; // Modifica o parâmetro dentro da função
}

let numero = 5;
adicionarDez(numero); // Chama a função com o valor do número
console.log(numero);  // O valor ainda é 5
```

Neste exemplo, a função **adicionarDez** recebe o valor de numero como parâmetro e o altera dentro da função, mas essa alteração não tem efeito fora dela. O valor original do parâmetro permanece inalterado porque ele é passado por valor.

Passagem por referência

Em contraste, se você passar um objeto ou array para uma função, a passagem é por referência. Nesse caso, a função recebe uma referência ao objeto original, não uma cópia. Alterações feitas no objeto dentro da função afetarão o objeto original fora da função. Veja um exemplo no quadro abaixo:

```
function alterarPropriedade(obj) {
    obj.nome = "Maria"; // Modifica a propriedade do objeto
}

let pessoa = { nome: "João" };
alterarPropriedade(pessoa); // Chama a função com uma referência ao objeto
console.log(pessoa.nome);  // O valor é alterado para "Maria"
```

Neste exemplo, a função **alterarPropriedade** recebe uma referência ao objeto **pessoa** e altera uma de suas propriedades. Como a função opera em uma referência ao objeto, a modificação é visível fora da função.

Valores padrão para parâmetros

A partir do ECMAScript 2015 (ES6), você pode definir valores padrão para parâmetros, que são usados se nenhum argumento for passado ao chamar a função.

Veja um exemplo no quadro abaixo:

```
function saudar(nome = "Visitante") {
    console.log("Olá, " + nome + "!");
}

saudar(); // Usa valor padrão, imprime "Olá, Visitante!"
saudar("Ana"); // Usa argumento passado, imprime "Olá, Ana!"
```

Os valores padrão são úteis para garantir que a função tenha um comportamento previsível mesmo quando argumentos são omitidos.

Parâmetros rest

Os parâmetros **rest** (rest parameters) permitem que uma função aceite um número indefinido de argumentos como uma lista de valores, representados como um array. Essa funcionalidade, introduzida no ECMAScript 2015 (ES6), é útil para criar funções que precisam lidar com uma quantidade variável de argumentos sem a necessidade de definir explicitamente todos os parâmetros.

Para usar parâmetros rest, você coloca três pontos (...) antes do nome do parâmetro. Isso indica que a função pode receber qualquer número de argumentos adicionais, que serão agrupados em um array.

Veja um exemplo no quadro abaixo:

```
function somar(...numeros) {
    return numeros.reduce((acc, cur) => acc + cur, 0); // Soma todos os valores
}

console.log(somar(1, 2, 3)); // Retorna 6
console.log(somar(5, 10, 15, 20)); // Retorna 50
```

Neste exemplo:

- A função **somar** usa o parâmetro rest **...numeros** para capturar todos os argumentos passados para a função.
- O método **reduce** é aplicado ao array numeros para somar todos os valores.
- A função **somar** pode aceitar qualquer quantidade de argumentos, de zero a muitos.

OBS: Os parâmetros rest devem ser o último elemento na lista de parâmetros da função. Isso garante que todos os argumentos restantes sejam capturados pelo parâmetro rest.

Veja um exemplo no quadro abaixo:

```
function mostrar(nome, ...outros) {
    console.log("Nome:", nome); // Primeiro argumento
    console.log("Outros:", outros); // Demais argumentos
}

mostrar("Alice", "Bob", "Charlie"); // "Nome: Alice", "Outros: ['Bob', 'Charlie']"
```

Neste exemplo, o primeiro argumento é tratado como um parâmetro normal, enquanto o restante é capturado pelo parâmetro rest outros.

◄ 3.2 Chamando funções

Seguir para...

3.4 Escopo ►