Var

Escopo de função ou global:

- Quando declarada dentro de uma função, a variável var só é acessível dentro dessa função.
- Quando declarada fora de qualquer função, ela se torna global, ou seja, acessível em todo o código.

Exemplo:

```
function exemplo() {
    var x = 10; // Escopo de função
    if (true) {
        var y = 20; // Mesmo dentro do bloco, 'y' é acessível fora dele
    }
    console.log(x); // 10
    console.log(y); // 20
}
exemplo();
console.log(x); // Erro: 'x' não está definido
console.log(y); // Erro: 'y' não está definido
```

Hoisting com undefined:

- A declaração da variável var é "elevada" (hoisted) para o topo do escopo, mas sua atribuição não.
- Isso significa que a variável existe, mas seu valor é undefined até que a linha de atribuição seja executada.

Exemplo:

```
console.log(nome); // undefined (hoisting)

var nome = "João";

console.log(nome); // João
```

Permite reatribuição e redeclaração:

- Você pode alterar o valor de uma variável var quantas vezes quiser.
- Também é possível redeclarar a mesma variável no mesmo escopo.

Exemplo:

```
var cor = "vermelho";

cor = "azul"; // Reatribuição permitida

var cor = "verde"; // Redeclaração permitida

console.log(cor); // verde
```

Let

Escopo de bloco:

- A variável let só é acessível dentro do bloco onde foi declarada (por exemplo, dentro de um if, for, ou {}).

  Exemplo:

```
function exemplo() {
    let x = 10; // Escopo de bloco (dentro da função)
    if (true) {
        let y = 20; // Escopo de bloco (dentro do if)
        console.log(y); // 20
    }
    console.log(x); // 10
    console.log(y); // Erro: 'y' não está definido
}
exemplo();
```

Hoisting sem inicialização:

- A declaração da variável let é elevada, mas ela não é inicializada. Tentar acessá-la antes da declaração resulta em um erro.

  Exemplo:

```
console.log(idade); // Erro: Cannot access 'idade' before initialization
let idade = 25;
console.log(idade); // 25
```

Permite reatribuição, mas não redeclaração:

- Você pode alterar o valor de uma variável let, mas não pode redeclarar a mesma variável no mesmo escopo.

  Exemplo:

```
let numero = 10;
numero = 20; // Reatribuição permitida
let numero = 30; // Erro: Identifier 'numero' has already been declared
```

Const

Escopo de bloco:

- Assim como let, a variável const só é acessível dentro do bloco onde foi declarada.

Exemplo:

```
function exemplo() {
    const x = 10; // Escopo de bloco (dentro da função)
    if (true) {
        const y = 20; // Escopo de bloco (dentro do if)
        console.log(y); // 20
    }
    console.log(x); // 10
    console.log(y); // Erro: 'y' não está definido
}
exemplo();
```

Hoisting sem inicialização:

- Assim como let, a declaração da variável const é elevada, mas ela não é inicializada. Tentar acessá-la antes da declaração resulta em um erro.

  Exemplo:

```
console.log(PI); // Erro: Cannot access 'PI' before initialization
const PI = 3.14;
console.log(PI); // 3.14
```

Não permite reatribuição nem redeclaração:

- Uma vez que um valor é atribuído a uma variável const, ele não pode ser alterado.
- Também não é possível redeclarar a mesma variável no mesmo escopo.

  Exemplo:

```
const nome = "Maria";
nome = "João"; // Erro: Assignment to constant variable
const nome = "Ana"; // Erro: Identifier 'nome' has already been declared
```

Permite alterar propriedades de objetos:

- Embora você não possa reatribuir uma variável const, é possível alterar as propriedades de um objeto ou os elementos de um array declarados com const.

Exemplo:

```
const pessoa = {
    nome: "Carlos",
    idade: 30
};
pessoa.idade = 31; // Alterando uma propriedade do objeto
console.log(pessoa); // { nome: 'Carlos', idade: 31 }

const numeros = [1, 2, 3];
numeros.push(4); // Adicionando um elemento ao array
console.log(numeros); // [1, 2, 3, 4]
```

Resumo de comparação:

Característica	**var**	**let**	**const**
Escopo	Função ou global	Bloco	Bloco
Hoisting	Levantada com valor undefined	Levantada, mas não inicializada	Levantada, mas não inicializada
Reatribuição	Permitida	Permitida	**Não permitida**
Redeclaração	Permitida	**Não permitida**	**Não permitida**
Uso com objetos	Pode alterar propriedades	Pode alterar propriedades	Pode alterar propriedades
Uso com primitivos	Pode alterar valor	Pode alterar valor	**Não pode alterar valor**

Quando usar cada um?

var:

- Evite usar var em código moderno, pois seu escopo de função e hoisting podem causar comportamentos inesperados.
- Use apenas se estiver trabalhando com código legado ou em situações específicas onde var é necessário.

let:

- Use let quando precisar de uma variável que pode ter seu valor alterado ao longo do código.
- \Ideal para loops, contadores ou variáveis que precisam ser reatribuídas.

const:

- Use const para valores que não devem ser reatribuídos.
- Ideal para constantes, configurações, objetos ou arrays que podem ter suas propriedades alteradas, mas não sua referência.

Exemplo prático combinando tudo:

```
// Exemplo de uso de var, let e const
function exemploCompleto() {
    var a = 10; // Escopo de função
    let b = 20; // Escopo de bloco
    const c = 30; // Escopo de bloco

    if (true) {
        var a = 50; // Redeclaração de 'a' (mesma variável)
        let b = 60; // Nova variável 'b' com escopo de bloco
        const c = 70; // Nova variável 'c' com escopo de bloco
        console.log(a); // 50
        console.log(b); // 60
        console.log(c); // 70
    }

    console.log(a); // 50 (alterado pelo bloco if)
    console.log(b); // 20 (não foi alterado pelo bloco if)
    console.log(c); // 30 (não foi alterado pelo bloco if)
}
exemploCompleto()
```