# VAR, LET, CONST

## Variable in JavaScript

Variables declared with `var` are hoisted to the top of their function scope, while variables declared with `let` and `const` are hoisted to the top of their block scope but remain uninitialized until their declaration In JavaScript, the `var` keyword is used to declare variables. Variables declared with `var` have function-level scope, meaning they are accessible throughout the function in which they are declared, including nested blocks. However, they are not block-scoped like variables declared with `let` or `const` in modern JavaScript.

```
var x = 10; // Declaring a variable named 'x' and assigning it the value 10


function myFunction() {
   var y = 20; // 'y' is accessible only within the scope of myFunction
   console.log(x); // Outputs: 10
   console.log(y); // Outputs: 20
}


console.log(x); // Outputs: 10
// console.log(y); // ReferenceError: y is not defined
```

- The variable `x` is declared using `var` and is accessible globally within the script because it's not inside any function or block.
- Inside the `myFunction` function, both `x` and `y` are accessible because `y` is declared using `var` within the function scope.
- However, `y` is not accessible outside the `myFunction` function because it's scoped to that function.

In JavaScript, the `let` keyword is used to declare block-scoped variables. Unlike variables declared with `var`, which have function-level scope, variables declared with `let` are scoped to the block (enclosed by curly braces) in which they are declared, including loops, conditional statements, and function blocks.

Here's how you can use `let` to declare variables:

```
let x = 10; // Declaring a variable named 'x' and assigning it the value 10


function myFunction() {
   let y = 20; // 'y' is accessible only within the scope of myFunction
   console.log(x); // Outputs: 10
   console.log(y); // Outputs: 20
}


console.log(x); // Outputs: 10
// console.log(y); // ReferenceError: y is not defined


{
   let z = 30; // 'z' is accessible only within this block
   console.log(z); // Outputs: 30
}


// console.log(z); // ReferenceError: z is not defined
```

- The variable `x` is declared using `let` and is accessible globally within the script because it's not inside any block.
- Inside the `myFunction` function, both `x` and `y` are accessible because `y` is declared using `let` within the function scope.
- The variable `z` is declared using `let` within a block, and it's accessible only within that block.

In JavaScript, the `const` keyword is used to declare constants, which are variables whose values cannot be reassigned or redeclared. Constants declared with `const` have block scope, similar to variables declared with `let`, meaning they are accessible only within the block in which they are defined.

Here's how you can use `const` to declare constants:

```
const PI = 3.14159; // Declaring a constant named 'PI' and assigning it the value 3.14159

console.log(PI); // Outputs: 3.14159

// Attempting to reassign a constant will result in an error
// PI = 3.14; // TypeError: Assignment to constant variable.

// However, the value of a constant object or array can be modified
const colors = ['red', 'green', 'blue'];
colors.push('yellow'); // Modifying the array is allowed
console.log(colors); // Outputs: ['red', 'green', 'blue', 'yellow']

// But reassigning the constant to a new object or array is not allowed
// colors = ['orange', 'purple']; // TypeError: Assignment to constant variable.
```

- The constant `PI` is declared using `const` and assigned the value `3.14159`. Once declared, the value of `PI` cannot be changed.
- Attempting to reassign a constant (`PI = 3.14;`) will result in a `TypeError`.
- However, if a constant holds an object or array, you can modify the properties or elements of the object/array. This doesn't violate the constant nature of `colors`, as it's the reference to the object that remains constant, not the object itself.

Interview Questions

1. **What is the difference between `var`, `let`, and `const` in JavaScript?**
   - `var` has function scope and can be redeclared and reassigned within its scope.
   - `let` has block scope and can be reassigned within its scope, but not redeclared.
   - `const` has block scope and cannot be reassigned or redeclared after initialization.
2. **When should you use `var`, `let`, or `const`?**
   - Use `var` for variables that need to have function scope or need to be accessible globally.

- Use `let` for variables that need to have block scope and may be reassigned.
- Use `const` for constants that should not be reassigned or redeclared after initialization.

3. **What is variable hoisting? How does it differ between `var`, `let`, and `const`?**
    - Variable hoisting refers to the behavior in JavaScript where variable declarations are moved to the top of their containing scope during the compilation phase.
    - is encountered.