Differences Between var and let

Summary: in this tutorial, you will learn about the differences between the `var` and `let` keywords.

#1: Variable scopes

The `var` variables belong to the global scope when you define them outside a function. For example:

```
var counter;
```

In this example, the `counter` is a global variable. It means that the `counter` variable is accessible by any function.

When you declare a variable inside a function using the `var` keyword, the scope of the variable is local. For example:

```
function increase() {
    var counter = 10;
}
// cannot access the counter variable here
```

In this example, the `counter` variable is local to the `increase()` function. It cannot be accessible outside of the function.

The following example displays four numbers from 0 to 4 inside the loop and the number 5 outside the loop.

```
for (var i = 0; i < 5; i++) {
        console.log("Inside the loop:", i);
}
```

```
console.log("Outside the loop:", i);
```

Output:

```
Inside the loop: 0
Inside the loop: 1
Inside the loop: 2
Inside the loop: 3
Inside the loop: 4
Outside the loop: 5
```

In this example, the `i` variable is a global variable. Therefore, it can be accessed from both inside and after the `for` loop.

The following example uses the `let` keyword instead of the `var` keyword:

```
for (let i = 0; i < 5; i++) {
        console.log("Inside the loop:", i);
}

console.log("Outside the loop:", i);
```

In this case, the code shows four numbers from 0 to 4 inside a loop and a reference error:

```
Inside the loop: 0
Inside the loop: 1
Inside the loop: 2
Inside the loop: 3
Inside the loop: 4
```

The error:

```
Uncaught ReferenceError: i is not defined
```

Since this example uses the `let` keyword, the variable `i` is blocked scope. It means that the variable `i` only exists and can be accessible inside the `for` loop block.

In JavaScript, a block is delimited by a pair of curly braces `{}` like in the `if...else` and `for` statements:

```
if(condition) {
   // inside a block
}

for(...) {
  // inside a block
}
```

#2: Creating global properties

The global `var` variables are added to the global object as properties. The global object is `window` on the web browser and `global` on Node.js:

```
var counter = 0;
console.log(window.counter); //  0
```

However, the `let` variables are not added to the global object:

```
let counter = 0;
console.log(window.counter); // undefined
```

#3: Redeclaration

The `var` keyword allows you to redeclare a variable without any issue:

```
var counter = 10;
var counter;
console.log(counter); // 10
```

However, if you redeclare a variable with the `let` keyword, you will get an error:

```
let counter = 10;
let counter; // error
```

#4: The Temporal dead zone

The `let` variables have temporal dead zones while the `var` variables don't. To understand the temporal dead zone, let's examine the life cycles of both `var` and `let` variables, which have two steps: creation and execution.

The var variables

- In the creation phase, the JavaScript engine assigns storage spaces to `var` variables and immediately initializes them to `undefined`.
- In the execution phase, the JavaScript engine assigns the `var` variables the values specified by the assignments if there are ones. Otherwise, the `var` variables remain undefined.

See the execution context for more information.

The let variables

- In the creation phase, the JavaScript engine assigns storage spaces to the `let` variables but does not initialize the variables. Referencing uninitialized variables will cause a `ReferenceError`.
- The `let` variables have the same execution phase as the `var` variables.

The temporal dead zone starts from the block until the `let` variable declaration is processed. In other words, it is the location where you cannot access the `let` variables before they are defined.

In this tutorial, you have learned about the differences between `var` and `let` keywords.