JavaScript Variables

A variable is a label that references a value like a number or string. Before using a variable, you need to declare it.

Declare a variable

To declare a variable, you use the **var** keyword followed by the variable name as follows:

```
var message;
```

A variable name can be any valid identifier. By default, the message variable has a special value undefined if you have not assigned a value to it.

Variable names follow these rules:

- Variable names are case-sensitive. This means that the message and Message are different variables.
- Variable names can only contain letters, numbers, underscores, or dollar signs and cannot contain spaces. Also, variable names must begin with a letter, an underscore (_) or a dollar sign ($).
- Variable names cannot use the reserved words.

By convention, variable names use camelcase like message, yourAge, and myName.

JavaScript is a dynamically typed language. This means that you don't need to specify the variable's type in the declaration like other static typed languages such as Java or C#.

Starting in ES6, you can use the let keyword to declare a variable like this:

```
let message;
```

It's a good practice to use the **let** keyword to declare a variable.

Initialize a variable

Once you have declared a variable, you can initialize it with a value. To initialize a variable, you specify the variable name, followed by an equals sign (=) and a value.

For example, the following declares the message variable and initializes it with a literal string "Hello":

```
let message;
message = "Hello";
```

To declare and initialize a variable at the same time, you use the following syntax:

```
let variableName = value;
```

For example, the following statement declares the message variable and initializes it with the literal string "Hello":

```
let message = "Hello";
```

JavaScript allows you to declare two or more variables using a single statement. To separate two variable declarations, you use a comma (,) like this:

```
let message = "Hello",
    counter = 100;
```

Since JavaScript is a dynamically typed language, you can assign a value of a different type to a variable. Although, it is not recommended. For example:

```
let message = 'Hello';
message = 100;
```

)

Change a variable

Once you initialize a variable, you can change its value by assigning a different value. For example:

```
let message = "Hello";
message = 'Bye';
```

Undefined vs. undeclared variables

It's important to distinguish between **undefined** and **undeclared** variables.

An undefined variable is a variable that has been declared but has not been initialized with a value. For example:

```
let message;
console.log(message); // undefined
```

In this example, the message variable is declared but not initialized. Therefore, the message variable is **undefined**.

In contrast, an undeclared variable is a variable that has not been declared. For example:

```
console.log(counter);
```

Output:

```
console.log(counter);
            ^
ReferenceError: counter is not defined
```

In this example, the counter variable has not been declared. Hence, accessing it causes a ReferenceError.

Constants

A constant holds a value that doesn't change. To declare a constant, you use the const keyword. When defining a constant, you need to initialize it with a value. For example:

```
const workday = 5;
```

Once defining a constant, you cannot change its value.

The following example attempts to change the value of the workday constant to 4 and causes an error:

```
workday = 2;
```

Error:

```
Uncaught TypeError: Assignment to constant variable.
```

Summary

- A variable is a label that references a value.
- Use the let keyword to declare a variable.
- An undefined variable is a variable that has been declared but not initialized while an undeclared variable is variable that has not been declared.
- Use the const keyword to define a read only reference to a value.

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 functions.

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: Life cycles of **var** and **let** variables

The life cycles of both **var** and **let** variables 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.

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.