ES6 Variables

A variable is a named space in the memory, which stores the values. The names of the variable are known as **identifiers.**There are some rules that you should keep in mind while naming an identifier. These rules are as follows:

* It can contain numbers and alphabets.
* You cannot start the name of the variable with a number.
* Keywords cannot be used as the name of the variable.
* Identifiers do not contain spaces and special characters except the **dollar ($) symbol**and **underscore (\_).**

Initialization of Variable

It is the process to store the value in the variable. A variable can be initialized at any time before its use.

The [ES6 syntax](https://www.javatpoint.com/es6-syntax) used the keyword **var** to declare a variable. In ES5, we declare the variable like this:

1. var x //Declaration of a variable by using the var keyword

Valid type of syntax for variable declaration

Some variable declarations are considered to be valid are as follows:

1. var $example1=value
2. var example1=value
3. var \_example$=value

In [ES6](https://www.javatpoint.com/es6), the variables are declared by:

* Using **let**.
* Using **const**.

let:

Any variable which is declared by using the **let**keyword is assigned the block scope. **Block scope**is nothing but a section where the **let**variable gets declared whether it is a **function{}**, a **block{}**, or a **global (script)**.

For example:**var v/s let**

**By using var**

1. var x = 100;
2. var x=200;
3. console.log(x);

When the code gets successfully executed, you will get the following output:

**Output:**

200

Let us try to re-write the above code by using the **let** keyword:

**By using let**

1. let x = 100;
2. let x=200;
3. console.log(x);

**Output:**

SyntaxError: Identifier 'x' has already been declared

When the code gets successfully executed, you will get an error that the identifier '**x**' has already been declared. So, any variable which is declared by using the **let** keyword is assigned the block scope.

const:

ES6 gives a new way to declare a constant using the **const** keyword. The keyword **const** creates a read-only reference to the value. There are some properties of the **const**that are as follows:

**Properties:**

* It cannot be reassigned a value.
* It is block scoped.
* A constant cannot be re-declared.
* Constants must be initialized at the time of declaration.

**For example:**

1. **const** y=100
2. y=200 // It will result in an error

**Output:**

TypeError: Assignment to constant variable.

It will throw an error because the constant variables are immutable and cannot be reassigned a value.

JavaScript Variable Scope

There are two scopes in [JavaScript](https://www.javatpoint.com/javascript-tutorial) that are **global** and **local:**

* **Global Scope:** In the global scope, the variable can be accessed from any part of the JavaScript code.
* **Local Scope:** In the local scope, the variable can be accessed within a function where it is declared.

**Example:**

The following example describes the Global and Local scope:

In this example, there are two variables one is outside the function (global scope), and the other is in the function (local scope).

1. var $var12 = 200;
3. function example() {
4. var $var12 = 300;
5. console.log("Inside example() function = "
6. + $var12);
7. }
8. console.log("Outside example() function = "
9. + $var12);
10. example();

**Output:**

Outside example() function = 200

Inside example() function = 300

JavaScript Dynamic Typing

JavaScript supports the concept of **Dynamic Typing** as similar to [python](https://www.javatpoint.com/python-tutorial), [perl](https://www.javatpoint.com/perl-tutorial), [ruby](https://www.javatpoint.com/ruby-tutorial), etc. It is a feature where you do not need to tell JavaScript about what type of value the variable will hold. If the type of value of the variable gets changed during the execution of the program, then it gets triggered, and JavaScript automatically takes care of it.

ES6 and Variable Hoisting

Hoisting is the default behavior of JavaScript to move all declarations to the top of the current script, current function, or current scope. It allows you to use the variable before its declaration. JavaScript only hoists the variable declaration, not variable initialization.

**For example:**

**JavaScript Declarations are Hoisted**

1. x=10;
2. console.log(x);
3. var x;

Instead of giving a declaration error, the above code will successfully execute and show the desired output. It happens because of the hoisting concept. Let us see what happens when the code is in the compiling phase.

When the above code is in the compile phase, then it will be treated as:

**In Compile phase**

1. var x; // declaration of the variable will move on top.
2. x=10;
3. console.log(x);

**Output:**

10

JavaScript Initializations are not Hoisted

**1. When you initialize the variable before using it**

1. var x=100;
2. var y=200;
3. console.log(x+"  "+y);

**In compiling phase**

1. var x;
2. var y;
3. x=100;
4. y=200;
5. console.log(x+"  "+y);

**Output:**

100 200

**II. When you initialize the variable after using it**

1. var x=100;
2. console.log(x+"  "+y);
3. var y=200;

Let us see what happens when this code is in the compile phase.

When this code is in compiling, then it will be treated as follows:

**In Compiling phase**

1. var x;
2. var y;
3. x=100;
4. console.log(x+"  "+y);
5. y=200;

When you execute this code, you will get the following output in which the value of **y** is undefined.

**Output:**

100 undefined