ES6 - Variables

A **variable**, by definition, is “a named space in the memory” that stores values. In other words, it acts as a container for values in a program. Variable names are called **identifiers**. Following are the naming rules for an identifier−

- Identifiers cannot be keywords.
- Identifiers can contain alphabets and numbers.
- Identifiers cannot contain spaces and special characters, except the underscore (_) and the dollar ($) sign.
- Variable names cannot begin with a number.

Type Syntax

A variable must be declared before it is used. ES5 syntax used the **var** keyword to achieve the same. The ES5 syntax for declaring a variable is as follows.

```
//Declaration using var keyword
var  variable_name
```

ES6 introduces the following variable declaration syntax −

- Using the let.
- Using the const.

Variable initialization refers to the process of storing a value in the variable. A variable may be initialized either at the time of its declaration or at a later point in time.

The traditional ES5 type syntax for declaring and initializing a variable is as follows −

```
//Declaration using var keyword
var variable_name = value
```

Example : Using Variables

```
var name = "Tom"
console.log("The value in the variable is: "+name)
```

The above example declares a variable and prints its value.

The following output is displayed on successful execution.

```
The value in the variable is Tom
```

JavaScript and Dynamic Typing

JavaScript is an un-typed language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically. This feature is termed as **dynamic typing**.

JavaScriptVariable Scope

The scope of a variable is the region of your program in which it is defined. Traditionally, JavaScript defines only two scopes-global and local.

- **Global Scope** − A variable with global scope can be accessed from within any part of the JavaScript code.
- **Local Scope** − A variable with a local scope can be accessed from within a function where it is declared.

Example : Global vs. Local Variable

The following example declares two variables by the name **num** - one outside the function (global scope) and the other within the function (local scope).

```
var num = 10
function test() {
   var num = 100
   console.log("value of num in test() "+num)
}
console.log("value of num outside test() "+num)
test()
```

The variable when referred to within the function displays the value of the locally scoped variable. However, the variable **num** when accessed outside the function returns the globally scoped instance.

The following output is displayed on successful execution.

```
value of num outside test() 10
value of num in test() 100
```

ES6 defines a new variable scope - The Block scope.

The Let and Block Scope

The block scope restricts a variable's access to the block in which it is declared. The **var** keyword assigns a function scope to the variable. Unlike the var keyword, the **let** keyword allows the script to restrict access to the variable to the nearest enclosing block.

```
"use strict"
function test() {
   var num = 100
   console.log("value of num in test() "+num) {
      console.log("Inner Block begins")
      let num = 200
      console.log("value of num : "+num)
   }
}
test()
```

The script declares a variable **num** within the local scope of a function and re-declares it within a block using the let keyword. The value of the locally scoped variable is printed when the variable is accessed outside the inner block, while the block scoped variable is referred to within the inner block.

Note − The strict mode is a way to opt in to a restricted variant of JavaScript.

The following output is displayed on successful execution.

```
value of num in test() 100
Inner Block begins
value of num : 200
```

Example: let v/s var

```
var no = 10;
var no = 20;
console.log(no);
```

The following output is displayed on successful execution of the above code.

```
20
```

Let us re-write the same code using the **let** keyword.

```
let no = 10;
let no = 20;
console.log(no);
```

The above code will throw an error: Identifier 'no' has already been declared. Any variable declared using the let keyword is assigned the block scope.

The const

The **const** declaration creates a read-only reference to a value. It does not mean the value it holds is immutable, just that the variable identifier cannot be reassigned. Constants are block-scoped, much like variables defined using the let statement. The value of a constant cannot change through re-assignment, and it can't be re-declared.

The following rules hold true for a variable declared using the **const** keyword −

- Constants cannot be reassigned a value.
- A constant cannot be re-declared.
- A constant requires an initializer. This means constants must be initialized during its declaration.

Example

```
const x = 10
x = 12 // will result in an error!!
```

The above code will return an error since constants cannot be reassigned a value. Constants variable are immutable.

ES6 and Variable Hoisting

The scope of a variable declared with var is its current execution context, which is either the enclosing function or, for variables declared outside any function, global. Variable hoisting allows the use of a variable in a JavaScript program, even before it is declared.

The following example better explains this concept.

Example: Variable Hoisting

```
var main = function() {
   for(var x = 0;x<5;x++) {
      console.log(x);
   }
   console.log("x can be accessed outside the block scope x value is :"+x);
   console.log('x is hoisted to the function scope');
}
main();
```

The following output is displayed on successful execution of the above code.

```
0
1
2
3
4
x can be accessed outside the block scope x value is :5
x is hoisted to the function scope
```

The JavaScript engine internally represents the script as −

```
var main = function() {
   var x; // x is hoisted to function scope
   for( x = 0;x<5;x++) {
      console.log(x);
   }
   console.log("x can be accessed outside the block scope x value is :"+x);
   console.log('x is hoisted to the function scope');
}
 main();
```

Note − The concept of hoisting applies to variable declaration but not variable initialization. It is recommended to always declare variables at the top of their scope (the top of global code and the top of function code), to enable the code resolve the variable’s scope.