Scopes, let & const

Scope in JavaScript

JavaScript has two types of scope:

- Block scope
- Function scope
- Global scope

Variables defined inside a function are in local scope, while variables defined outside of a function are in the global scope. Each function, when invoked, creates a new scope.

Previously before ES6, JavaScript had only Global Scope and Function Scope.
ES6 introduced variable declaration methods like let, const, which now provides block scope also.

Block Scope

Variables declared inside a { } block cannot be accessed from outside the block.
To provide a block scope, you can't use the **var** keyword; either use the let or const keyword.

```
Example : {
                var temp = "Coding Ninjas";
          }
          console.log(temp); // Output coding Ninjas
```

But,

```
Example : {
                let temp = "Coding Ninjas";
          }
          console.log(temp); // Not Accessible outside the block
```

Function

A JavaScript function is a block of code designed to perform a particular task.
A JavaScript function is executed when it is called.

Basic Syntax :

```
function function_name( ){
          //code
}
```

Functions will be covered in-depth in the upcoming modules.

Local Scope

Variables declared within a JavaScript function becomes **local variables** to the function and have a local scope.

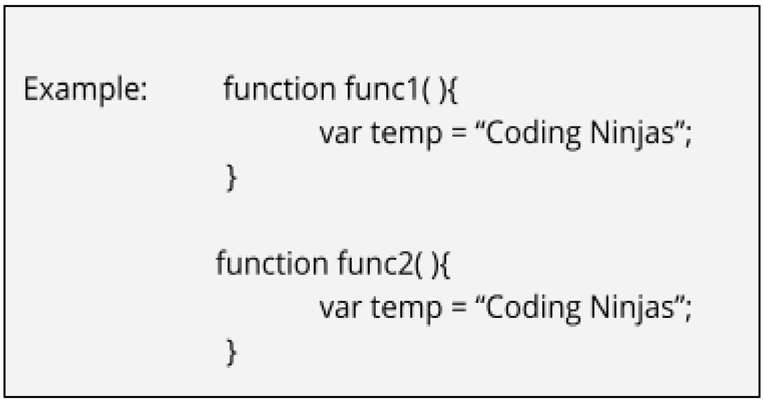
```
Example:   //  temp is NOT accessible outside the function

              function function_name( ) {
                var temp = "Coding Ninjas";
                // temp is accessible within this function
              }

            //  temp is NOT accessible outside the function
```

In the above example, the temp variable had a local scope to the function, and it can only be accessed within that function.

Variables with the same name in different functions can also be declared
Variables made inside a function are moved to the memory when the function is called and deleted from the memory as soon as the function completes its work

Function Scope

Variables declared inside the function are only accessible within that function. Declaring two variables with the same name in different functions is allowed and gives no error.



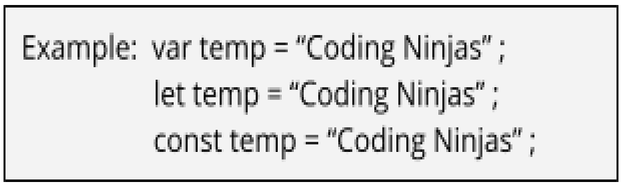
```
Example:    function func1( ){
                  var temp = "Coding Ninjas";
            }

            function func2( ){
                  var temp = "Coding Ninjas";
            }
```

In the above example, temp was declared two times. Each had different scope for the respective function in which it was declared.

Variables declared with **var**, **let,** and **const** are quite similar when declared inside a function.

Global scope

- ➔ Variable declared outside a function have global scope.
- ➔ Global variables can be accessed from anywhere in a JavaScript program.
- ➔ Variables declared with **var, let and const** act as same if declared globally.



```
Example:  var temp = "Coding Ninjas" ;
          let temp = "Coding Ninjas" ;
          const temp = "Coding Ninjas" ;
```

All of the above are globally declared variables and can be accessed anywhere in the program.

Block scope vs Function scope

Variables declared with the var keyword are global if declared within a block, but the declaration is made in a function. It can only be accessed within that function.

```
Example : {
            var temp = "Coding Ninjas";
        }
        console.log(temp) ; // Output Coding Ninjas
But ,
      function fun( ){
            var temp = "Coding Ninjas";
            }
      console.log(temp) ; // ERROR
```

In the first case, temp was a global variable, but in the second case, it was a function variable.

Hoisting of var

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.
Visualise it like this:

```
console.log (temp);
var temp= "Coding Ninjas" ;
```

JavaScript interprets it as :

```
var temp; //Declaration
console.log(temp);
temp = "Coding Ninjas"; // Definition
```

So var variables are hoisted to the top of their scope and initialised with a value of undefined.

We'll study the undefined later in the module.

Let

let is now preferred for variable declaration. It's no surprise as it comes as an improvement to var declarations. It also solves the problem with var that it is globally available.

Properties of let :

➔ **let is { } block scoped**

```
Example : {
                let temp = "Coding Ninjas";
          }
          console.log(temp); // Not Accessible outside the block
```

➔ Declarations of let variables are also hoisted, but we cannot use them before its variable definition. The block of code is aware of the variable, but it cannot be used until it has been declared.

```
Example :  console.log(temp);
           let temp= "Coding Ninjas" ;
```

Shows a Reference Error

➔ **Let can be updated but not re-declared**: Just like var, a variable declared with let can be updated within its scope. Unlike var, a let variable cannot be re-declared within its scope.

```
let temp= "Coding Ninjas";
temp= "Coding Ninjas Coding" ;
```

Updation is allowed but

```
let temp = "Coding Ninjas" ;
let temp = "Coding Ninjas Coding" ;
```

Re-declaration is NOT allowed.

Const

Variables declared with the const maintain constant values. Const declarations share some similarities with let declarations.

Properties of const:

➔ **const declarations are also block scoped**. Like let declarations, const declarations can only be accessed within the block they were declared.

➔ **Mandatory variable definition :**

INCORRECT :

```
const temp ;
temp = “Coding Ninjas” ;
```

CORRECT :

```
const temp = “Coding Ninjas” ;
```

➔ **It cannot be updated or re-declared**

INCORRECT :

```
const temp= "Coding Ninjas" ;
temp= "Coding” ; // error: Assignment to constant variable.
```

INCORRECT :

```
const temp= "Coding Ninjas" ;
const temp= "Coding” ; // error: Identifier 'greeting' has already been declared
```

➔ **Hoisting of const**: Just like let, const declarations are hoisted to the top but are not initialised.

Differences and Key-Points

- var declarations are globally scoped, or function scoped, while let and const are blocks scoped.
- var variables can be updated and re-declared within its scope; let variables be updated but not re-declared; const variables can neither be updated nor re-declared.
- They are all hoisted to the top of their scope. But while var variables are initialised with undefined, let, and const variables are not initialised.
- While var and let can be declared without being initialised, const must be initialized during declaration.