1. **Variable Scope** :

Variable scope in JS is function level and not block level like most of the programming languages.

for (var i = 1; i <= 10; i++) {

console.log (i); // outputs 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;

};

// The variable i is a global variable and it is accessible in the following function with the last value it was assigned above

function aNumber () {

alert(i);

}

// The variable i in the aNumber function below is the global variable i that was changed in the for loop above. Its last value was 11, set just before the for loop exited:

aNumber (); // 11

Example 2

// Both firstName variables are in the global scope, even though the second one is surrounded by a block {}. ​

​var firstName = "Richard";

{

​var firstName = "Bob";

}

​// To reiterate: JavaScript does not have block-level scope​

​// The second declaration of firstName simply re-declares and overwrites the first one​

console.log (firstName); // Bob

1. Function Hoisting:

JS moves function declaration to top. This make us use a function call before its declaration.

But function hoisting only applies to named functions.

Function expressions don't get hoisted.

Reason : Function declarations load before any code is executed.

Function expressions load only when the interpreter reaches that line of code.

alert(foo()); // ERROR! foo wasn't loaded yet

var foo = function() { return 5; }

alert(foo()); // Alerts 5. Declarations are loaded before any code can run.

function foo() { return 5; }

1. Does hoisting applies to variables?

All variable declarations are hoisted (lifted and declared) **to the top of the function**, if defined in a function, or the top of the global context, if outside a function.

1. **setTimeout Variables are Executed in the Global Scope**  
   Note that all functions in setTimeout are executed in the global scope. This is a tricky bit; consider this:

// The use of the "this" object inside the setTimeout function refers to the Window object, not to myObj​

​

​var highValue = 200;

​var constantVal = 2;

​var myObj = {

highValue: 20,

constantVal: 5,

calculateIt: function () {

setTimeout (function () {

console.log(this.constantVal \* this.highValue);

}, 2000);

}

}

​

​// The "this" object in the setTimeout function used the global highValue and constantVal variables, because the reference to "this" in the setTimeout function refers to the global window object, not to the myObj object as we might expect.​

​

myObj.calculateIt(); // 400​

​// This is an important point to remember.

1. It is important to know that only variable declarations are hoisted to the top, not variable initialization or assignments (when the variable is assigned a value.

function showName () {

console.log ("First Name: " + name);

​var name = "Ford";

console.log ("Last Name: " + name);

}

​

showName ();

​// First Name: undefined​

​// Last Name: Ford​

​

​// The reason undefined prints first is because the local variable name was hoisted to the top of the function​

​// Which means it is this local variable that get calls the first time.​

​// This is how the code is actually processed by the JavaScript engine:​

​

​function showName () {

var name; // name is hoisted (note that is undefined at this point, since the assignment happens below)​

console.log ("First Name: " + name); // First Name: undefined​

​

name = "Ford"; // name is assigned a value​

​

​// now name is Ford​

console.log ("Last Name: " + name); // Last Name: Ford​