Variable Scoping

IF variable is defined but not initialized then it will displayed undefined

If variable is not defined but used in code then it will display reference error exception

Hoisting is the JavaScript interpreter’s action of moving all variable and function declarations to the top of the current scope. However, only the actual declarations are hoisted. Any assignments are left where they are.

Before

(function() {

var foo = 1;

var bar = 2;

var baz = 3;

alert(foo + " " + bar + " " + baz);

})();

After

(function() {

var foo;

var bar;

var baz;

foo = 1;

bar = 2;

baz = 3;

alert(foo + " " + bar + " " + baz);

})();

Function Hoisting

function declarations are also hoisted but function expression are not

This will work as it is function declaration

foo();

function foo() {

alert("Hello!");

}

This will fail as it is function expression

foo();

var foo = function() {

alert("Hello!");

};

JavaScript has two scopes – global and local. Any variable declared outside of a function belongs to the global scope, and is therefore accessible from anywhere in your code. Each function has its own scope, and any variable declared within that function is only accessible from that function and any nested functions. Because local scope in JavaScript is created by functions, it’s also called function scope. When we put a function inside another function, then we create nested scope.

Currently, JavaScript, unlike many other languages, does not support block level scoping. This means that declaring a variable inside of a block structure like a for loop, does not restrict that variable to the loop. Instead, the variable will be accessible from the entire function. It is supported by **let** keyword in ES6

In JavaScript, variables with the same name can be specified at multiple layers of nested scope. In such case local variables gain priority over global variables. If you declare a local variable and a global variable with the same name, the local variable will take precedence when you use it inside a function. This type of behavior is called shadowing. Simply put, the inner variable shadows the outer.

<https://www.sitepoint.com/demystifying-javascript-variable-scope-hoisting/>

* Functions are hoisted first, and then variables.
* Function declarations have priority over variable declarations, but not over variable assignments.

<http://www.adequatelygood.com/JavaScript-Scoping-and-Hoisting.html>

**Why global scope is bad??**

All scopes in JavaScript are created with Function Scope only, they aren’t created by for or while loops or expression statements like if or switch.

Whenever you see a function within another function, the inner function has access to the scope in the outer function, this is **called Lexical Scope or Closure** - also referred to as Static Scope.

The only important thing to remember is that Lexical scope does not work backwards

// name = undefined

var scope1 = function () {

// name = undefined

var scope2 = function () {

// name = undefined

var scope3 = function () {

var name = 'Todd'; // locally scoped

};

};

};

Scope refers to the visibility of variables and context refers to the value of this in the same scope

<https://stackoverflow.com/questions/500431/what-is-the-scope-of-variables-in-javascript>

 Module Pattern

(function () {

var myFunction = function () {

// do some stuff here

};

})();

myFunction(); // Uncaught ReferenceError: myFunction is not defined

myFunction is private scoped

var Module = (function () {

return {

myMethod: function () {

console.log('myMethod has been called.');

}

};

})();

Module.myMethod() is public function

var Module = (function () {

var privateMethod = function () {

//Contains addClass/AJAX calls,helpers etc.

};

return {

publicMethod: function () {

// has access to `privateMethod`, we can call it:

// privateMethod();

}

};

})();

<http://javascriptissexy.com/javascript-variable-scope-and-hoisting-explained/>

<https://scotch.io/tutorials/understanding-scope-in-javascript>

<https://developer.mozilla.org/en/docs/Web/JavaScript/Closures>

<http://javascriptissexy.com/understand-javascript-closures-with-ease/>

**Overriding setTimeout in JS**

var temp = setTimeout;

var setTimeout = function() {};

**First Class Function : Function passed as parameter to another function / function expression**

**Var greetMe = function(){**

**Console.log(‘hi’);**

**}**

**Function greet(fn){**

**Fn();**

**}**

**Greet(greetMe);**

Node.js is a C++ program controlled via the Chrome V8 engine, which is used as a standalone process (detached from Chrome) and whose job is to compile javascript to native code and execute it.

**libuv** is a multi-platform C library that provides support for asynchronous I/O based on event loops.

Src/win/core.c

Buffer – A temporary holding spot for data.

Stream – Sequence of data available over time.

Readable,Writable,Duplex,Transform,PassThrough streams