Module 3 – Vanilla JavaScript: Scope, Closure and Hoisting

# Lab Objectives

Explore what is meant by Scope in JavaScript

See practical examples of closure

Create real world examples of hoisting

Scope

Scope in JavaScript can be either local or global, every variable we have created so far has had global scope and can be accessed from anywhere in the code we have written, however variables can also be declared within the functions we write, these will be blocked off from the rest of the application and cannot be accessed outside of the function.

Task: fix the following code and write another function utilising local scope variables

let exampleNumberOne = 1;

let exampleNumberTwo = 4;

function add(numOne) {

let result = exampleNumberTwo + exampleNumberOne;

}

console.log(result);

Closure

Closure in JavaScript means we can write concise code. Each function has access to the variables within its local scope, outer function scope, and at the global scope.

Task: answer the questions about the following code:

Is the variable randNum accessible at a global level?

let exampleNumberOne = 1;

let exampleNumberTwo = 4;

let result;

function addTwoNumsPlusRandom(numOne, numTwo) {

function getRandomNum() {

let randNum;

randNum = Math.floor((Math.random() \* 10) + 1);

return randNum;

}

result = numOne + numTwo + getRandomNum();

}

console.log(result);

Can the function getRandomNum access exampleNumberOne or exampleNumberTwo?

Where can the function getRandomNum be called from?

Task: Write another piece of code including a use of closure

Closures can store references to their outer functions variables, this means that even though you may just be returning an object or function, it still has access to the variables at the level above itself. An example of this is shown below:

function userID() {

let userID = 15;

return {

getUserID: function () {

return userID;

},

setUserID: function (newUserID) {

userID = newUserID;

}

};

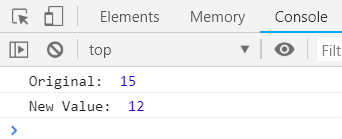
}

let userIdObject = userID();

console.log('Original: ', userIdObject.getUserID());

userIdObject.setUserID(12);

console.log('New Value: ', userIdObject.getUserID());



The object returned from userID still has a reference to the userID property defined in the outer function, allowing us to return or edit the value.

Task: Write another piece of code utilising a closure in this way

Hoisting

Due to a process called hoisting we can use variables in JavaScript before they are defined. This is because when the code is running all variable declarations are moved to the very top of the file by default, and are the first thing to be executed. While the declaration of variables is moved to the top of the file, the assignment of values to them is not. An example of hoisting is shown below:

x = 5;

console.log(x);

var x;

//--------------------

var x = 5;

console.log(x);

Both sections of this code produce the same result, because the use of the var declation means the definition of x is moved to the top of the file.

**Task:** In the following code explain why the top example will print undefined, and the bottom example will print the number 5

console.log(x);

var x = 5;

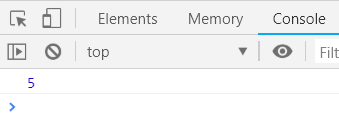
//----------------------

x = 5;

console.log(x);

var x;

Hoisting is also applicable to functions, we can use functions above their definition in the code because this is automatically moved to the top of the file. An example of this is shown below:



printNumFive();

function printNumFive() {

console.log(5);

}

**Task:** Explain why the following code examples will, or will not work

let exampleId = 6;

console.log('Original ', exampleId);

addOneToNumber(exampleId);

function addOneToNumber(number) {

console.log('with 1 added: ', number + 1);

}

//----------------------------------------------------------------------

let exampleId = 6

console.log('Original ', exampleId);

addOneToNumber(exampleId);

let addOneToNumber = function(number) {

console.log('with 1 added: ', number + 1);

}

//----------------------------------------------------------------------

let exampleId = 6

console.log('Original ', exampleId);

addOneToNumber(exampleId);

let addOneToNumber = (num) => console.log('with 1 added: ', num + 1);

//-----------------------------------------------------------------------

let exampleId = 6

addOneToNumber = function (number) {

console.log('with 1 added: ', number + 1);

}

console.log('Original ', exampleId);

addOneToNumber(exampleId);

var addOneToNumber;