## **COURSEWARE**

Professional Skills	
Agile Fundamentals	
Jira	
Git	
Databases Introduction	
Java Beginner	
Maven	
Testing (Foundation)	
Java Intermediate	
HTML	
CSS	
Javascript	
0	What is JavaScript
0	Getting started with JS
0	Variables
0	Data types
0	ASI
0	Strict mode
0	Iteration
0	Conditionals with Truthy / Falsey
0	Objects, Arrays + JSON
0	Structuring JS Code
0	Destructuring
0	Scope
0	Functions, function expressions and arrow functions
0	The ECMAScript 6 Specification
0	OOP in JavaScript
0	Best Practices
0	Closures
0	Callbacks and Promises
0	Cookies
0	Hoisting
0	Prototypes
0	Query Parameters

Higher Order Functions

# Scope

#### Contents

- Overview
- Tutorial
  - Accessibility
    - Function Scope
    - Global Scope
    - Automatically Global
    - Working from the inside outward
      - Example 1
      - Example 2
    - Rememeber
- Exercises

## Overview

In this module we will explore scope within javascript.

In JavaScript there are TWO main types of scope:

- Local Scope
- Global Scope

Scope determines accessibility / visibility of these variables.

## **Tutorial**

## Accessibility

Variables that are declared inside a function **aren't** accessible / visible outside the function.

Trying to reference a variable outside of it's function will cause a Reference Error!

```
function someFunction() {
  let hello = "Billy Bob Joe";
}
someFunction();
console.log(hello); // ReferenceError: hello is not defined
```

## **Function Scope**

Local variables have **function scope** meaning that it only exists within the function that it is created in! We cannot access the variable outside of the function nor alter it outside the function!

See **above** example

## Global Scope

Variables declared outside a function have **global scope** - it exists and can be accessed throughout the whole file including inside functions.

For example:

Web Storage DOM Manipulation Handling Events and Timed Events Asynchronous Programming HTTP-Requests **XMLHttpRequests** Fetch API Spring Boot Selenium Sonarqube Advanced Testing (Theory) Cucumber MongoDB **Express NodeJS** React **Express-Testing** Networking Security Cloud Fundamentals **AWS Foundations AWS Intermediate** Linux DevOps Jenkins Introduction

Jenkins Pipeline

IDE Cheatsheet

Markdown

```
let check = false;
 function changeCheck() {
   check = true;
 changeCheck();
 console.log(check); // True

    We can access the value of check inside a function and alter it

   • The value will be updated across the file given that it has global scope.
Automatically Global
If we don't have a variable declared in the file but it is referenced inside a
function without a lettable keyword then the variable is automatically added to
the global scope!
This can be very dangerous as the value can be altered anywhere within the
file now.
 function someFunction() {
   hello = "Billy Bob Joe";
 }
 someFunction();
 console.log(hello); // Billy Bob Joe
 hello = "Who is Billy?";
 console.log(hello); // Who is Billy?
```

## Working from the inside outward

When we are referencing variables we always look for a **local variable** first inside our function, if there is none then we look for a **global variable**, and if none exists then one is added to the global scope!

Take a look at the following code:

## Example 1

```
function test() {
  flag = true;
  alert(flag);
  test1();
  alert(flag);
}
function test1() {
  flag = false;
  return;
}
```

- 1. First we invoke test();
- 2. We declare a variable called flag and set the value to true it doesn't contain the keyword let/const therefore we assume it is a **global variable**
- 3. Alert the value of flag --> true
- 4. Invoke and drop into test1();
- 5. Update the value of flag to false again it has global scope
- 6. Return to previous position in test();
- 7. Alert the value of flag --> false.

#### Example 2

```
function test() {
  flag = true;
  alert(flag);
  test1();
  alert(flag);
}
function test1() {
  let flag = false;
  return;
}
```

- 1. Invoke test();
- 2. Check for a local variable there isn't one so we use the *global variable* of flag set to true;
- 3. Alert the value of flag true;
- 4. Invoke and drop into test1();
- 5. A local variable flag exists in test1() with the value of false
- 6. Return to previous test();
- 7. Alert the value of flag which is still true

We DIDN'T update the global value of flag in test1(), we created a local variable inside the function but remember local variables can't be read outside its function scope therefore the GLOBAL variable is still the same!

#### Rememeber

- Scope defines how variables can be seen / accessed
- Use the let keyword to specify scope to the current block
- If you don't use let then the variable has global scope
- Unless you declare a variable in a function or block it is of global scope
- Scope chains define how an identifier is looked up start from the inside and work out
  - Check if there is a local variable IF NOT then check if there is a global variable
- If there is no local or global variable then one is added to the global scope!

## **Exercises**

- 1. Write the following code and assess the output
  - Create a function
  - Declare a variable with a value inside it (i.e. let x = 'foo')
  - Write an if statement that checks if the variable meets a condition
  - Inside create a local variable
  - Try to access both variables and asses your output
  - ► Solution
- 2. What is the result of executing this code and why?

```
function doSomething() {
  console.log(a);
  console.log(foo());
  let a = 1;
  function foo() {
    return 2;
  }
}
doSomething();
```

▶ Solution