CS472 WAP

JavaScript Scope

Scope, Closures, and Encapsulation

Except where otherwise noted, the contents of this document are Copyright 2012 Marty Stepp, Jessica Miller, Victoria Kirst and Roy McElmurry IV. All rights reserved. Any redistribution, reproduction, transmission, or storage of part or all of the contents in any form is prohibited without the author's expressed written permission.
Slides have been modified for Maharishi University of Management Computer Science course CS472 in accordance with instructors agreement with authors.

Maharishi International University
Fairfield, Iowa © 2020

All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi International University.

Main Point Preview

- JavaScript has global scope and local scope within functions when variables are declared with var, and now has block scope with const and let.

- *Science of Consciousness: Local and global scopes are similar to fine and broad awareness. The experience of transcending opens our awareness to the expanded scope of unbounded awareness, at the same time that it promotes the ability to focus sharply within any local boundaries.*

The global object

- technically no JavaScript code is "static" in the Java sense
  - all code lives inside of some object
  - there is always a `this` reference that refers to that object

- all code is executed inside of a global object
  - in browsers, it is also called window;
  - global variables/functions you declare become part of it
    - they use the global object as this when you call them

- "JavaScript's global object [...] is far and away the worst part of JavaScript's many bad parts." -- D. Crockford

Implied globals

```
function foo() {
  x = 4;
  print(x);
} // oops, x is still alive now (global)
```

- if you assign a value to a variable without var, JS assumes you want a new global variable with that name
  - Any typo becomes an undetected ‘bug’ !!
  - this is a "bad part" of JavaScript (D.Crockford)

Scope

- scope: The enclosing context where values and expressions are associated.
  - essentially, the visibility of various identifiers in a program

- lexical scope: Scopes are nested via language syntax; a name refers to the most local definition of that symbol.
  - most modern languages (Java, C, ML, Scheme, JavaScript)

- dynamic scope: A name always refers to the most recently executed definition of that symbol. It searches through the dynamic stack of function calls for a variable declaration.
  - Perl, Bash shell, Common Lisp (optionally), APL, Snobol
  - See slide 24 (Scope Example 1)

Lexical scope in Java

- In Java, every block ({}) defines a scope.

```
public class Scope {
    public static int x = 10;

    public static void main(String[] args) {
        System.out.println(x);
        if (x > 0) {
            int x = 20;
            System.out.println(x);
        }
        int x = 30;
        System.out.println(x);
    }
}
```

Lexical scope in JavaScript (pre-ES6)

- In JavaScript, there are (were) only two scopes:
  - global scope: global environment for functions, vars, etc.
  - function scope: every function gets its own inner scope

Global Scope

Function Scope

```
var x = 10;
function main() {
  console.log("x1: " + x);
  x = 20;
  if (x > 0) {
    var x = 30;
    console.log("x2: " + x);
  }
  var x = 40;
  var f = function(x) { console.log("x3: " + x); }
  f(50);
}
main();
```

Lack of block scope

```
for (var i = 0; i < 10; i++) {
  console.log("i inside for loop: " + i);
}
console.log(i); // 10
if (i > 5) {
  var j = 3;
}
console.log("j OUTside block: " + j);
```

- any variable declared lives until the end of the function
  - lack of block scope in JS leads to errors for some coders
  - this is a "bad part" of JavaScript (D. Crockford)

var vs let (ES6)

- var scope – nearest function scope
- let scope – nearest enclosing block

```
function a() {
  for (var x = 1; x < 10; x++) {
    console.log(x);
  }
  console.log("x: " + x);
  //10
}
```

→

```
function a() {
  for (let x = 1; x < 10; x++) {
    console.log(x);
  }
  console.log("x: " + x);
  //ReferenceError: x is not defined
}
```

- Use let inside for loops to prevent leaking to Global Scope
  - never use var in new JS code

Best Practices

- variables defined with `var` are hoisted and have value `undefined` until it is assigned a value in code
  - Do not use var assignments in new code
- When using `let` or const, there will be no hoisting and we will receive a reference `error` if used before they are declared
- Best practice is to use const or let and explicitly declare them before using
  - Makes code more obvious for humans to understand
  - Use `const` by default
  - Only use `let` if you need to update variable later
  - Don’t use `var`
- But, millions of legacy programs use var and any competent JS programmer must understand hoisting

Best Practice: const, let, var, looping

"I personally recommend you always use const, as it leads to fewer bugs. I have yet to encounter a situation where I needed to use var. As a general rule, use let only for loop counters or only if you really need reassignment. Everywhere else, use const.

Personally, I’ve ditched loops for filter(), map() & reduce(). You should too."

(freecodecamp.org blog)

Main Point

- JavaScript has global scope and local scope within functions when variables are declared with var, and now has block scope with const and let.

- *Science of Consciousness: Local and global scopes are similar to fine and broad awareness. The experience of transcending opens our awareness to the expanded scope of unbounded awareness, at the same time that it promotes the ability to focus sharply within any local boundaries.*


MAHARISHI
HIGHER EDUCATION
FOR HIGHER CONSCIOUSNESS
INTERNATIONAL UNIVERSITY