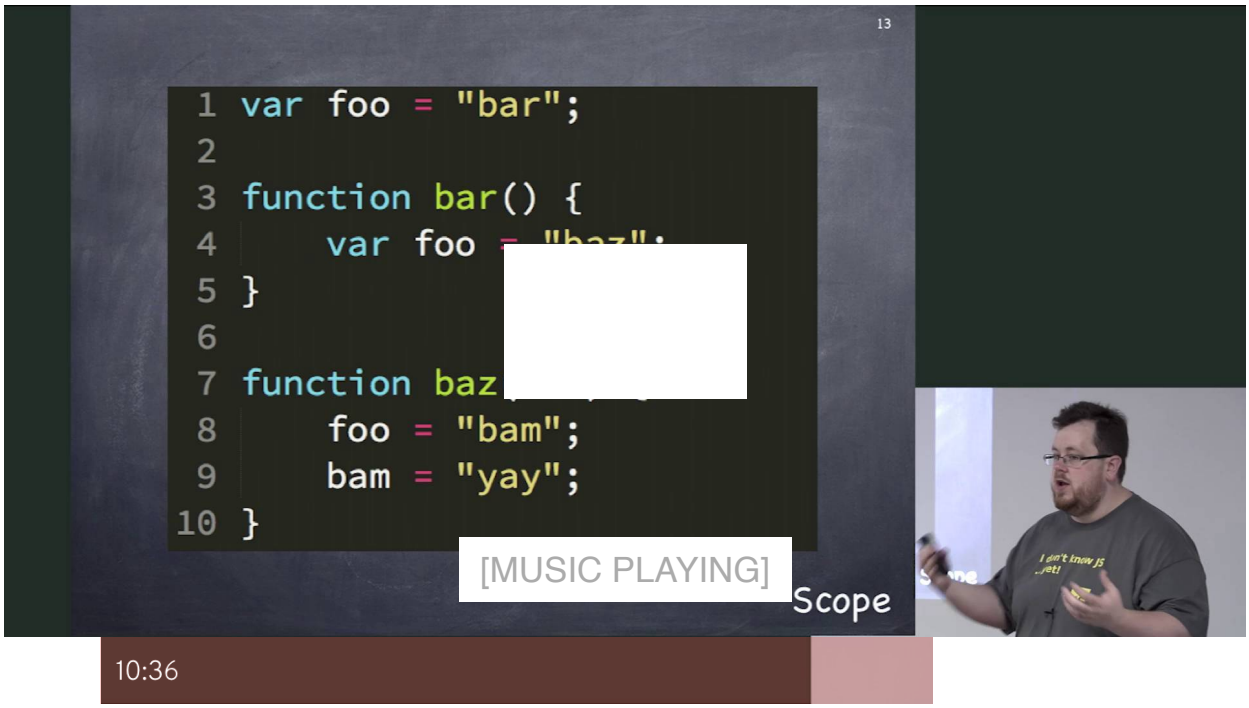
Advanced JavaScript

Kyle Simpson
You Don't Know JS

6 hours, 51 minutes **CC**



This course has been updated and renamed to Deep JavaScript Foundations! Go watch that course instead.
Kyle Simpson, author of the "You Don't Know JavaScript" book series, details the inner workings of JavaScript in extreme detail. Gain an advanced understand of the core mechanics of the JavaScript language. This course deep dives into scope, closure, object-oriented, and asynchronous programming in JavaScript.

This course and others like it are available as part of our Frontend Masters video subscription.

Published: June 14, 2014

Table of Contents

Scope and Execution Example 01:11:30 - 01:24:43

Kyle walks the audience through another example of how the JavaScript compiler will declare and execute variables and functions. This example includes a nested function which creates a nested scope.

Function Declarations, Function Expressions, and Block Scope 01:24:44 - 01:34:38

A function declaration occurs when the function keyword is the first word of the statement. Functions assigned to a variable become function expressions. Kyle explains these difference while also describing why it is bad to use anonymous functions.

Lexical Scope 01:34:39 - 01:39:39

Both Lexical Scope and Dynamic Scope are two models of scope programming languages. Meaning "compile-time scope," Kyle uses a building metaphor to help explain Lexical Scope.

Cheating Lexical Scope: eval 01:39:40 - 01:48:50

Since there are many ways to cheat in JavaScript, Kyle demonstrates how the "eval" keyword can be used to cheat Lexical Scope rules. He also describes issues that arise when using the "with" keyword.

IIFE Pattern 01:48:51 - 01:58:22

The Immediately Invoked Function Expressions (IIFE) Pattern is a technique used to hide scope involving wrapping code inside a function that is immediately called. This technique allows developers to create an object in their scope without polluting the outer scope.

IIFE Pattern Questions 01:58:23 - 02:02:40

Before discussing the let keyword, Kyle fields a few questions about syntax style with the IIFE pattern.

Block Scope in ES6 02:02:41 - 02:06:52

In ECMAScript 6, the "let" keyword will implicitly create a block-level scope and add declarations to that scope rather than the enclosing function. The most common use-case for the let keyword is for loops.

Problems with let keyword 02:06:53 - 02:15:03

Kyle describes a few issues he has with the let keyword. Some of his issues are stylistic, but others are related to common variable functionality like hoisting. Kyle discusses his solutions for these issues and a tool he created to help. - http://github.com/getify/let-er

Dynamic Scope 02:15:04 - 02:16:55

Using a hypothetical example, Kyle briefly describes dynamic scope as it relates to Lexical scope.

zz: Scope 02:16:56 - 02:18:39

Kyle presents a quiz about what was covered in the Scope section of this course and reviews the answers with the audience.

Hoisting 02:18:40 - 02:31:35

Hoisting is the moving of declarations to the top of the scope block during the compiling phase. Hoisting applies to both variable declarations and functions. Kyle spends some time explaining why hoisting exists in JavaScript and the problems surrounding it.

Exercise 1 02:31:36 - 02:34:30

For this exercise, you will be using the files in the day1/ex1 folder. Look at the README.md file for exercise instructions.

Exercise 1: Solution 02:34:31 - 02:42:26

Kyle walks through the solution for exercise 1.

this Keyword 02:42:27 - 02:54:31

Every function, while it's executing, has a reference to its current execution context called "this." The "this" reference is JavaScript's version of dynamic scope. Kyle explains the "this" keyword and its relationship to the call site of the function.

Binding Confusion 02:54:32 - 03:01:07

Attempting to force the "this" keyword into a different lexical scope can lead to some binding confusion. Kyle pulls an example he found on Stack Overflow around this confusion to further demystify usage of the "this" keyword.

Explicit Binding 03:01:08 - 03:13:47

The explicit binding rule allows developers to use the "call" method and pass an explicit reference for the "this" binding. Explicit bindings can also be set using the "apply" method. Kyle explains explicit bindings and also detours into a discussion about a technique he calls "hard binding." Hard binding was added as of ES5 in the form of the "bind" method.

The new keyword 03:13:48 - 03:21:59

JavaScript does not have classes and the "new" keyword does not do any instantiation. Kyle explains the functionality of the "new" keyword and the effects it has when placed in front of a function call.

Quiz: this 03:22:00 - 03:25:31

Kyle presents a quiz about the "this" keyword.

Closure

Closures 03:25:32 - 03:32:18

Closures are often misunderstood by JavaScript developers. Closures are when a function remembers its lexical scope even when the function is executed outside that lexical scope.

Closure Examples 03:32:19 - 03:40:16

To further explain closure, Kyle shows examples using some common JavaScript structures like setTimeout and click events. He also demonstrates closure in shared scopes and nested scopes.

More Closure Examples 03:40:17 - 03:45:38

Kyle demonstrates a few additional closure examples inside loops and the misconceptions that arise. He also compares closure to traditional object references to explain the difference.

Module Patterns 03:45:39 - 03:56:08

Kyle explains the different module patterns that use closure. This includes the classic, modified, and modern patterns. He also discusses what to expect in ES6.

Quiz: Closure 03:56:09 - 04:00:09

Kyle presents a quiz about the different closure topics he covered.

Exercise 2 04:00:10 - 04:02:08

For this exercise, you will be using the files in the day1/ex2 folder. Look at the README.md file for exercise instructions.

Exercise 2: Solution 04:02:09 - 04:12:14

Kyle walks through the solution for exercise 2.

Object-Oriented

Prototype 04:12:15 - 04:17:20

In JavaScript, every object is built by a constructor function and does not mean classes are being instantiated. When a constructor function is called, a new object is created with a link to the object's prototype.

Prototypes Explained, Part 1 04:17:21 - 04:27:24

Using a code example from the slides, Kyle spends some time diagramming the relationship between an object and its prototype.

Prototypes Explained, Part 2 04:27:25 - 04:34:22

Kyle explains the relationship between __proto__ (dunder-proto) and the prototype keyword and how both reference the underlining prototype. ES5 added a standardized way to do this using the getPrototypeOf method.

Prototype Linkages 04:34:23 - 04:42:54

Prototype linkages allow delegation to other objects to hand method calls or property references. This allows additional objects to be created from a prototype with duplication of the function code. This binding is beneficial as long as developers don't break any rules.

prototype: Objects Linked 04:42:55 - 04:49:11

Prototypes in JavaScript can be linked and share a parent-child relationship similar to a subclass and superclass. This is beneficial when extending a prototype to add additional methods. However, there are issues with constructor references.

Linked Prototype Diagram 04:49:12 - 04:53:39

Kyle revisits the prototype diagram he drew on the whiteboard earlier. This time, however, he shows a more complex version outlining the relationship of the two linked prototypes.

Quiz: Object Prototypes 04:53:40 - 04:56:42

Kyle presents a quiz about object prototypes and their behavior.

Exercise 3 04:56:43 - 04:59:32

For this exercise, you will be using the files in the day1/ex3 folder. Look at the README.md file for exercise instructions.

Exercise 3: Solution 04:59:33 - 05:06:17

Kyle walks through the solution for exercise 3.

Inheritance 05:06:18 - 05:11:24

In classical inheritance, properties and methods of a class are copied to object instantiated of that class. Subclasses inherit the properties and methods of a parent class and copy them to their instantiated objects. Kyle contrasts that with JavaScript's "prototypal inheritance" or "behavior delegation"

OLOO 05:11:25 - 05:17:43

Rather than relating prototypes to inheritance, Kyle demonstrates that prototypes allow actions to be delegated to other objects. Kyle refers to this a Objects Linked to Other Objects or OLOO. He modifies the previous example to use this OLOO technique.

OLOO Questions 05:17:44 - 05:26:51

Now that Kyle has explained this OLOO method for creating and delegating objects, he spends a few minutes answering audience questions. He also compares the old prototype-based code with the new OLOO code and shows all the prototype functionality is moved to the Object.create method. - http://gist.github.com/getify/5572383 - http://gist.github.com/getify/5226305

Quiz: Prototype Unit 05:26:52 - 05:34:06

Kyle presents a quiz about the prototype unit.

Exercise 4 05:34:07 - 05:36:18

For this exercise, you will be using the files in the day1/ex4 folder. Look at the README.md file for exercise instructions. In the interest of time, Kyle gives this exercise to the audience as a "homework" assignment.

Exercise 4 Solution 05:36:19 - 05:58:37

Kyle spends a few minutes at the beginning of the next day describing the solution to exercise 4.

Async Patterns

Callbacks 05:58:38 - 06:06:53

Callbacks are integral to JavaScript but can lead to many problems. They allow for asynchronicity to occur, but in the process, create an inversion of control where developers are handing off control of their application to another area or mechanism.

Solving Callback Problems 06:06:54 - 06:09:56

Kyle demonstrates a few techniques developers have used to get around callback issues. For example, providing separate callbacks in the case of success and failure functionality. Most of these solutions only lead to more inversion of control.

Generators 06:09:57 - 06:17:30

Generators are coming in ES6. A generator is a new type of function that can be paused during execution and resumed at a later time. The are paused using the yield keyword.

Promises 06:17:31 - 06:25:22

Kyle explains how promises are a way to subscribe to the completion of a task. For example, when a function is called, a promise will let you know when the function completes. Kyle demonstrates jQuery's implementation of promises and compares it to the native implementation in ES6.

asynquence 06:25:23 - 06:31:11

Kyle shows a library he wrote as an alternative to promises called asynquence. Sequences are automatically chained promises. His library, asynquence, represents asynchronous sequences. - http://github.com/getify/asynquence/

Quiz: Async Patterns unknown

Exercise 5 06:32:07 - 06:34:38

For this exercise, you will be using the files in the day1/ex5 folder. Look at the README.md file for exercise instructions. In the interest of time, Kyle gives this exercise to the audience as a "homework" assignment.

Exercise 5 Solution 06:34:39 - 06:54:24

Kyle spends a few minutes at the beginning of the next day describing the solution to exercise 5.

Email: support@frontendmasters.com
Phone: +1 (612) 324-1071

Twitter Facebook