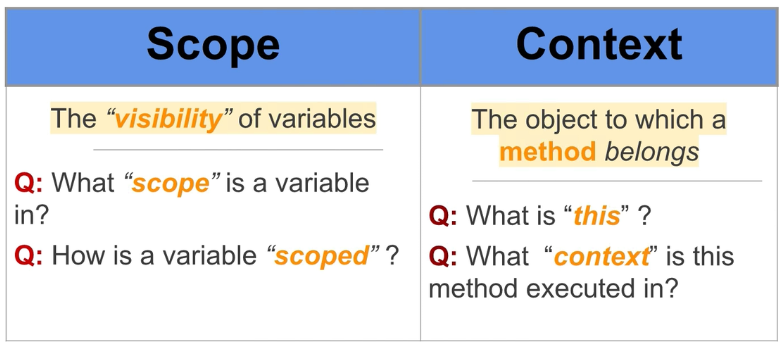
[https://www.skillshare.com](https://www.skillshare.com/)

**Scope and Context**



When you think of scope – Think of variables

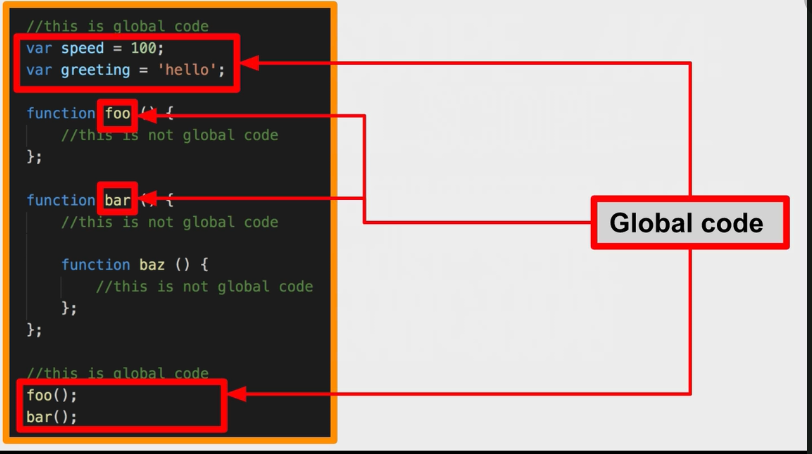
When you think of context – Think of methods

When it comes to context, “this” keyword is really important!

1. SCOPE

**Global Scope**

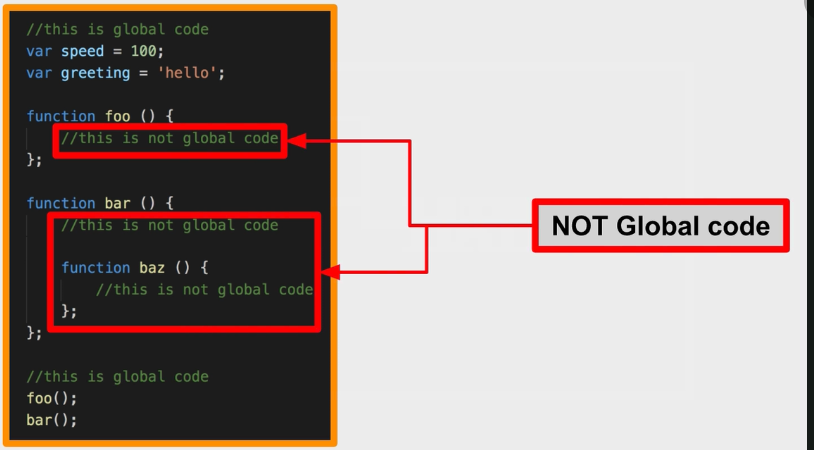
Global scope is anything that is not in a function.



Here, speed, greeting, foo and bar are property of window object.

Global Scope aka :

1. Window Object
2. Outermost scope
3. Top Level scope

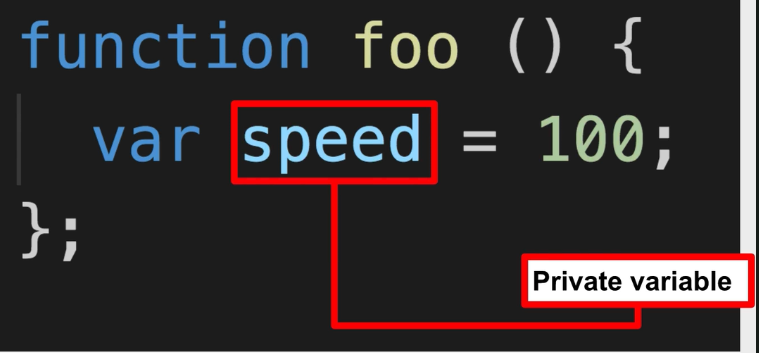


**Private scope**

Private scope is achieved by using “var” keyword inside of a function. “var” keyword is like “private” in Java, but here in JavaScript it is creating variables private to a function unlike class in Java.

AKA,

1. Local Scope
2. Function Scope

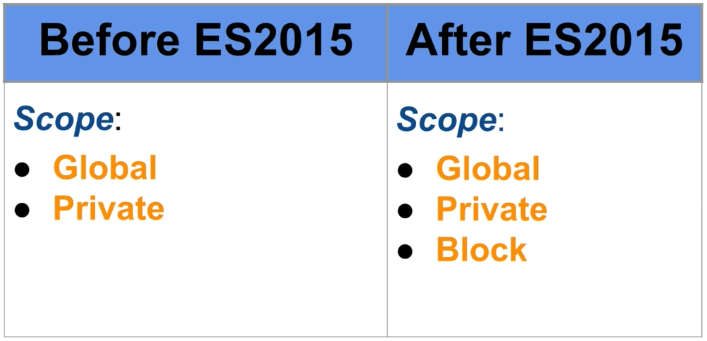




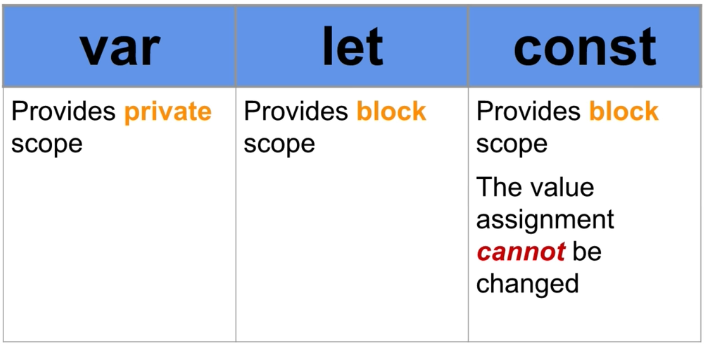
Lexical Scope is a way scope behaves in JavaScript.

**Block Scope**

Block scope is achieved by using Let or Const keyword inside of a block { }. Block scope was introduced in ECMA Script 2015.

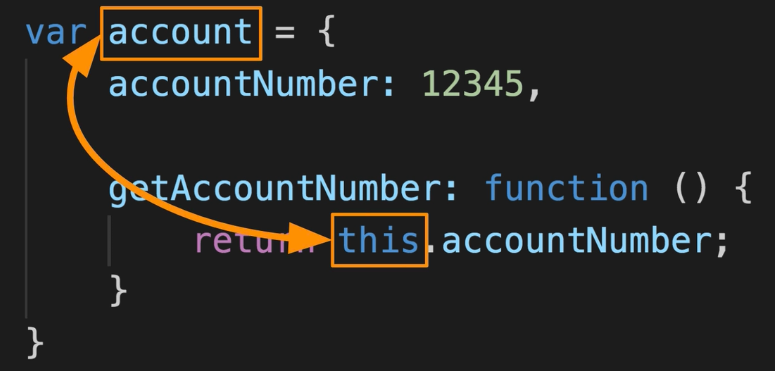


Three different ways to create a variable:



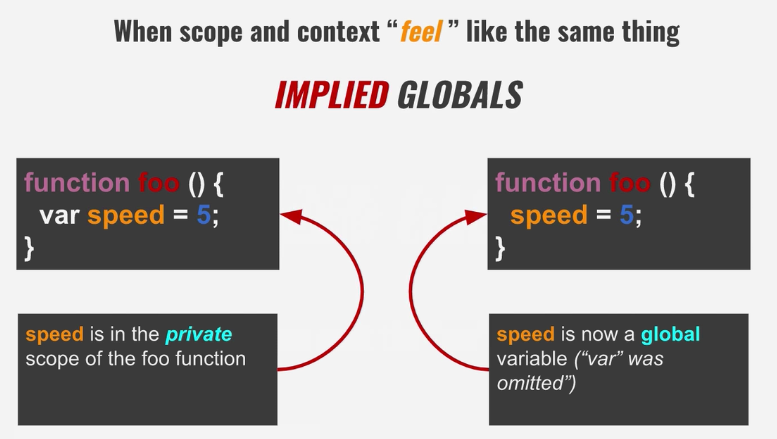
1. CONTEXT

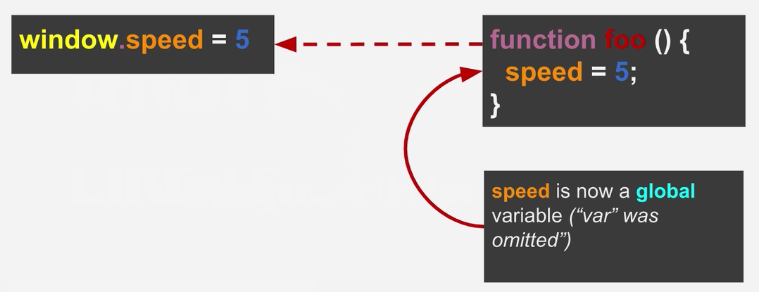
In JavaScript, when a property is a function, then it is called a METHOD.

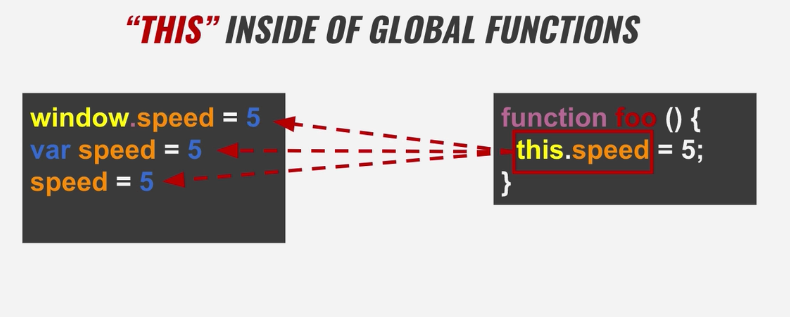


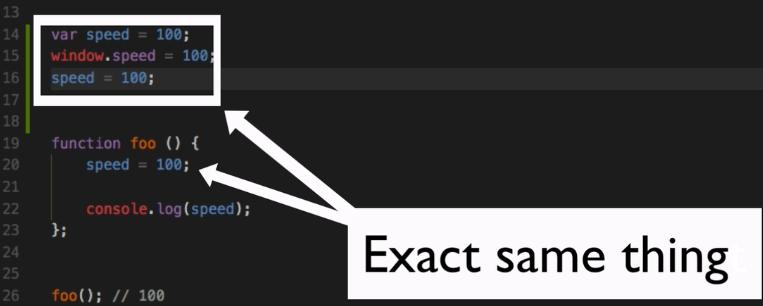
Here, getAccountNumber property is a method.

Sometimes it happens that context and scope feel like same thing. But actually they are NOT.







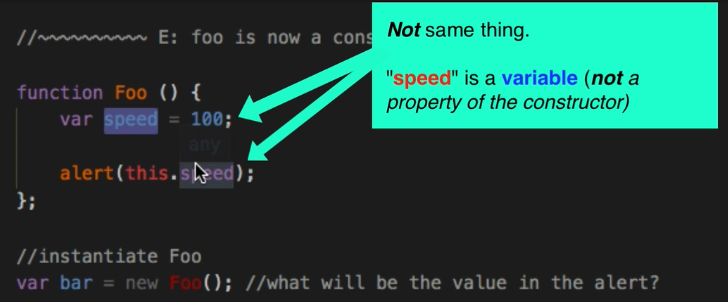


The declaration outside foo() is Explicit declaration of global variable

The declaration inside of foo() is Implicit declaration of global variable

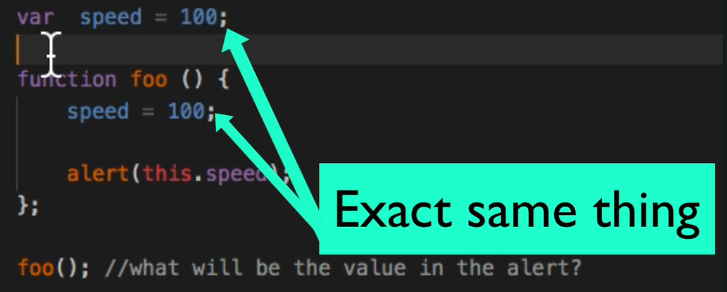


When we are in strict mode, we cannot access the window object using “this” from a global function.



The output: is “undefined”

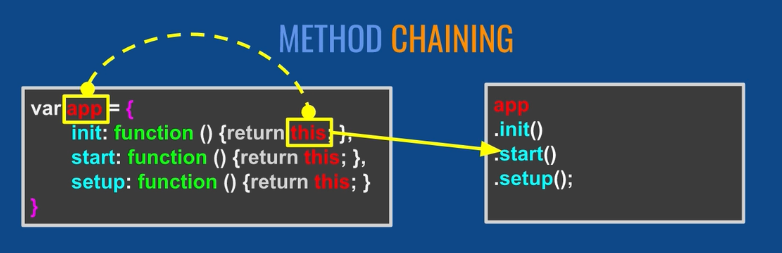


Above, alert will give 100.Since omitted var keyword while defining speed inside of foo(), we created a global speed variable.

SWITCHING CONTEXT

This is achieved by using call(), apply() and bind() functions. They provide the ability to change the object which a method belongs in other words, it’s like changing value if “this” for a particular function.

METHOD CHAINING



Here, every method is returning “this” which in turn means “app”. So we can call methods on each other because it will be equivalent to calling on “app” object.

App.init().start().setup() < ==equivalent to== > app.init(); app.start(); app.setup();

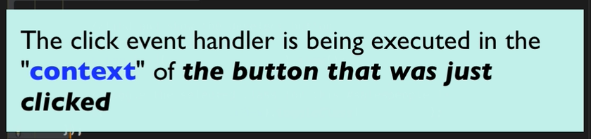
This is called Method chaining. It does not actually changes the way your code works, but it does allows you to write that is easier to read and little bit more expressive.

App.init().start().setup() < == This statements tells me that all these methods are returning “app” object.

TROUBLESHOOTING: SCOPE PROBLEM OR CONTEXT PROBLEM

If the problem is because of a variable then it is a scope problem, and if it is because of this keyword then it is a context problem.

Once you get to know what type your problem is, next thing you do as part of debugging process is to console.log(variable\_name) in case of scope problem and console.log(this) in case of context problem.



This means that “this” inside of a click event handler points to the DOM element which is clicked and not the object inside of which the event handler is defined.