JS and jQuery Jubilee

This shouldn’t be you

Remember this!

"You can't tell whether you're learning something when you're learning it—in fact, learning feels a lot more like frustration."

"What I've learned is that during this period of frustration is actually when people improve the most, and their improvements are usually obvious to an outsider. If you feel frustrated while trying to understand new concepts, try to remember that it might not feel like it, **but you're probably rapidly expanding your knowledge.**"

Jeff Dickey, Author of Write Modern Web Apps with the MEAN Stack: Mongo, Express, AngularJS, and Node.JS

Feedback #1 – Pace is Fast!!!

- That said, as instructors / TAs we are here to help.
- As we fall into a class rhythm, feel encouraged to schedule a 1-1 during office hours.
- In addition to using the time to understand concepts… it's a great way for us to identify weaknesses and outline steps to get on the right track.
- These might be before / after class.

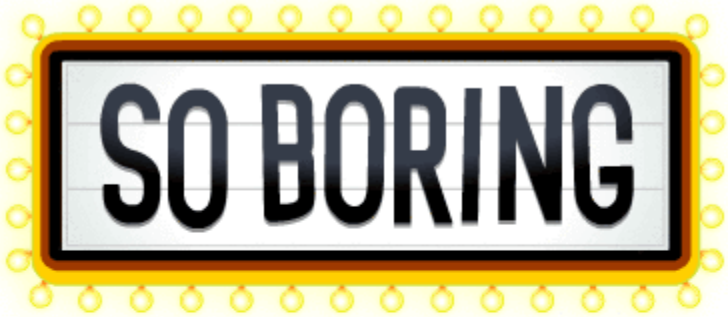
Today’s Class

Objectives

1. **Pretend to learn scoping**
2. **Build a jQuery Calculator**

Lexical Scope

Shh… Just Between Us.



<u>WARNING:</u>

This next section is heavy on theory.

Disclaimer:

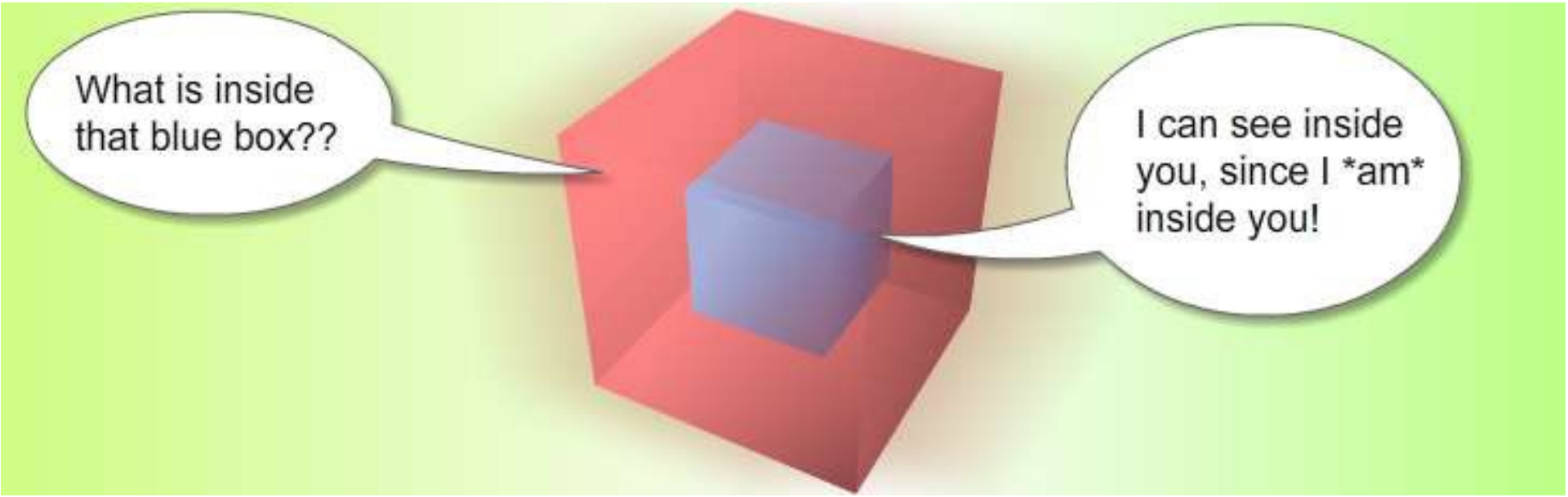
It’s not the end of the world if its confusing and/or you’re completely lost.

Javascript Scope

- In Javascript curly brackets { } indicate blocks of code.
- In order for the code inside the curly brackets to be executed, it must meet the condition or it must be called (example: functions).
- These blocks of code have the power to affect variables outside the curly brackets if those variables were declared outside – so be careful!

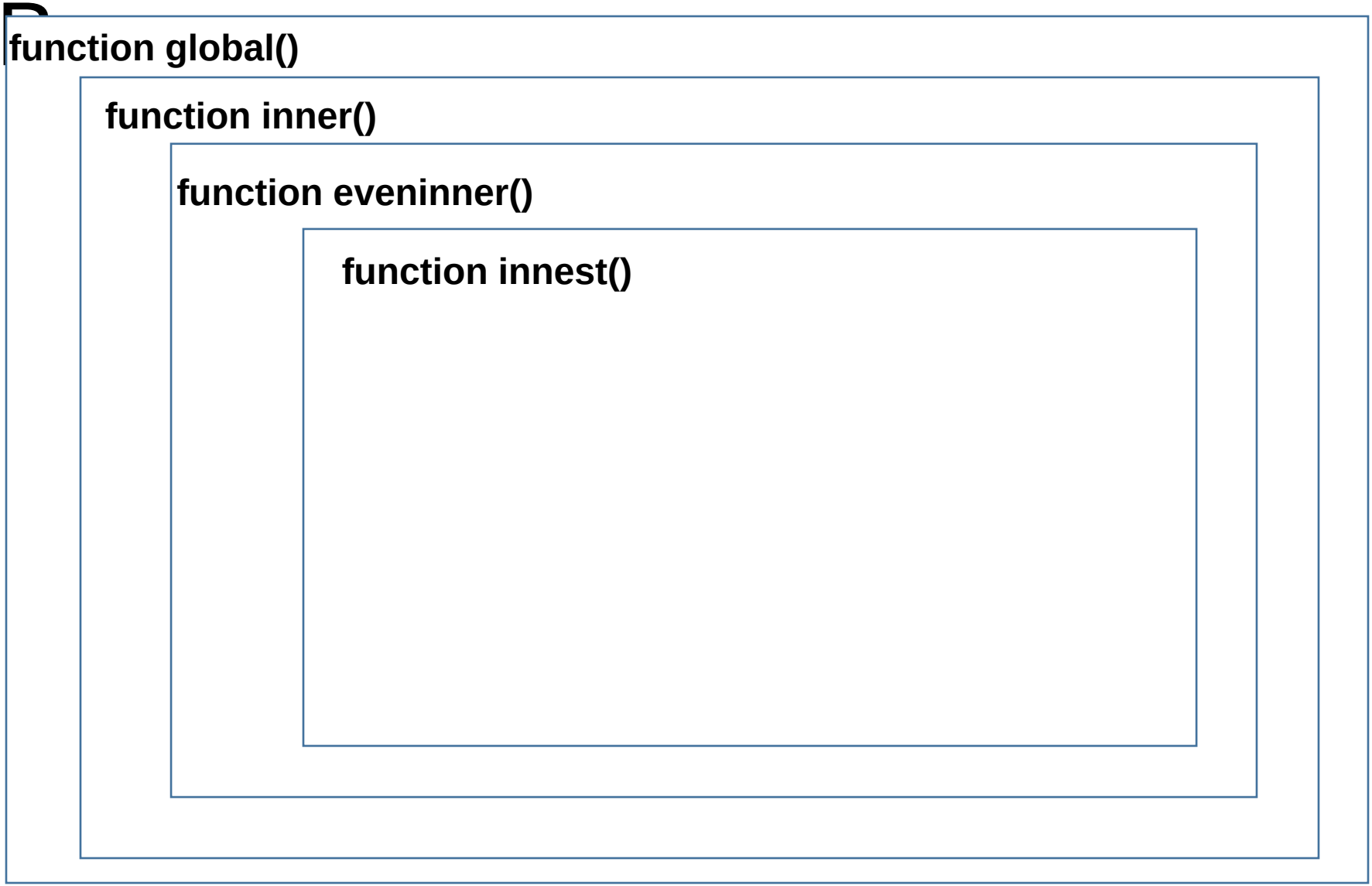
```
// Sets initial value of x
var x = 5;

// False Condition doesn't get run
if(1 > 2000) {
    x = 10
}

// Will print 5. X was unchanged.
console.log(x);
```

Scope = Boxes in Boxes

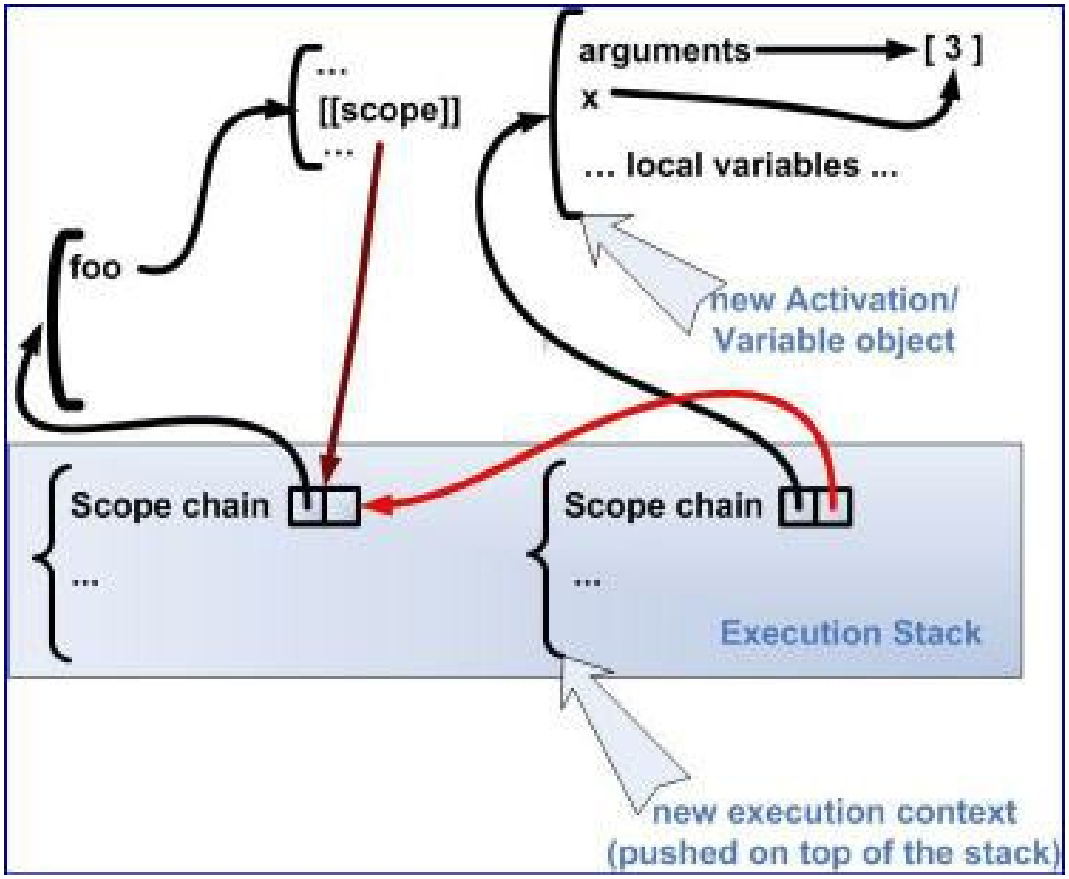


Scope impacts which variables can be accessed by which function.

Scope = Boxes in P



Javascript's Odd Relationship with Scope



For those who have programmed in other languages, Javascript seemingly behaves in unpredictable ways.

Javascript Scope Example (Tricky)

```
<script>
  var outerFunction = function() {

    var x = 5;

    var nestedFunction = function() {

      var y = 7;

      // What will this print? (x = 5)
      console.log("X: " + x);

      // What will this print? (y = 7)
      console.log("Y: " + y);

      var z = 10;
      // What will this print? (z = 10)
      console.log("Z (inside): " + z);
    };

    return nestedFunction;
  };

  var myFunction = outerFunction();
  myFunction();

  // What will this print? (z is undefined)
  console.log("Z (outside): " + z);

</script>
```

Here **nested function** is clearly able to access the variables of their **parent function**.

Whereas **outer function** has no idea what the variable z is because it was declared in a child function.

> YOUR TURN!!

Activity: *14-ScopeOne* | **Suggested Time:** 5 min

- Take a few moments dissecting what I just said.
- Look at the file sent to you and explain to the person next to you what is meant by:
  - The terms parent function and child function
  - The concept that child functions can access parent variables but not vice versa.
- **Be prepared to share!**

> YOUR TURN!!

Activity: *15-ScopeTwo* | **Suggested Time:** 7 min

- Take a few moments to dissect the code just sent to you.
- Try to predict what will be printed in each of the examples.
- **Be prepared to share!**
- Note: Pay attention to the unusual use of the keyword: ‘this”

> YOUR TURN!!

Activity: *16-ScopeThree* | **Suggested Time:** 7 min

- Take a few moments to dissect the code just sent to you.
- Try to predict what will be printed in each of the examples.
- **Be prepared to share!**
- Note: Pay attention to the unusual use of the keyword: ‘this”

> YOUR TURN!!

Activity: *17-ScopeQuiz* | **Suggested Time:** 7 min

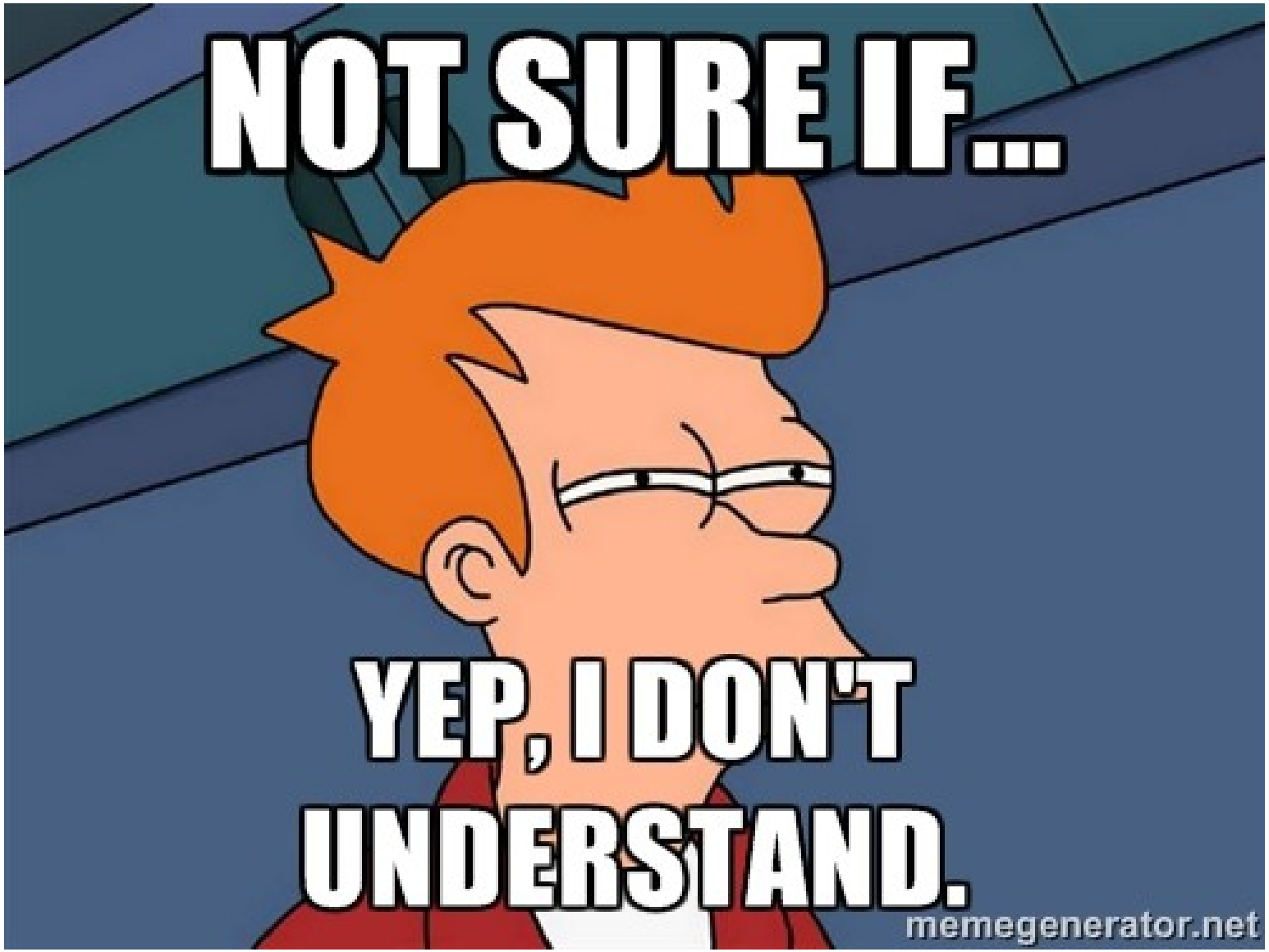
- Spend a few moments studying the codefile with the person sitting next to you.
- Then run the program in the browser.
- Once you run the program, you'll find that Code Block 1 leads to different alerts than Code Block 2.
- Ask your partner which Code Block is behaving the way you would expect.
- Then work with your partner to try and identify the specific difference that is causing the issue with the faulty block.
- Once you spot the issue, try to explain to your partner why JavaScript is handling these Code Blocks differently.

> YOUR TURN!!

Activity: *18-ThisExample* | **Suggested Time:** 10 min

- Using the comments in the guide answer each of the questions asked in the file.
- Focus your attention on trying to wrap your mind around the concept of "this" and the unique role it can play in code.
- Then run the program in the browser.
- Then try to explain to your partner how "this" works, focus on the first three examples.

You Probably…



Helpful Article (If you'd like to learn more…)

< back to Blog Home

BY: KAITLIN DAVIS

POSTED IN:
- Web Apps

October 20, 2014 by: Kaitlin Davis

9 Comments

What You Should Already Know about JavaScript Scope

If you are a novice JavaScript programmer, or if you've been messing around with JQuery to pull off a few animations on your website, chances are you're missing a few vital chunks of knowledge about JavaScript.

One of the most important concepts is how scope binds to "*this*".

For this post, I'm going to assume you have a decent understanding of JavaScript's basic syntax/objects and general terminology when discussing scope (block vs. function scope, this keyword, lexical vs. dynamic scoping).

Lexical Scoping

First off, JavaScript has *lexical scoping* with *function scope*. In other words, even though JavaScript looks like it should have block scope because it

Questions