jQuery Jubilee

Web Development Boot Camp

Lesson 5.2

This shouldn’t be you:

Admin Items

- Welcome Will!
- Homework 4: due Tuesday Sept 29

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*

Important Reminders

This course covers a lot of material quickly, so remember:

- Instructors and TAs are here to help.
- Feel encouraged to schedule a one-on-one during office hours.
- One-on-one sessions are a great way to identify weaknesses and outline a plan to get back on track.
- Office hours are held before and after class.

Today’s Class

Objectives

01 Use jQuery DOM manipulation to create simple games.

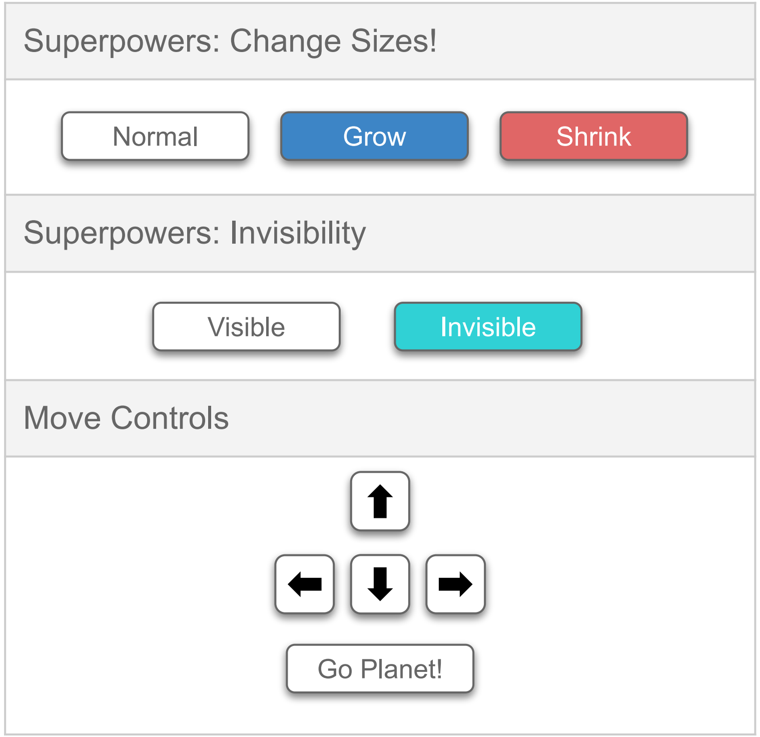
02 Practice jQuery on Captain Planet: The Game and Fridge Game.

03 Gain an initial understanding of lexical scope in JavaScript.

04 Understand click events.

Captain Planet: The Game!

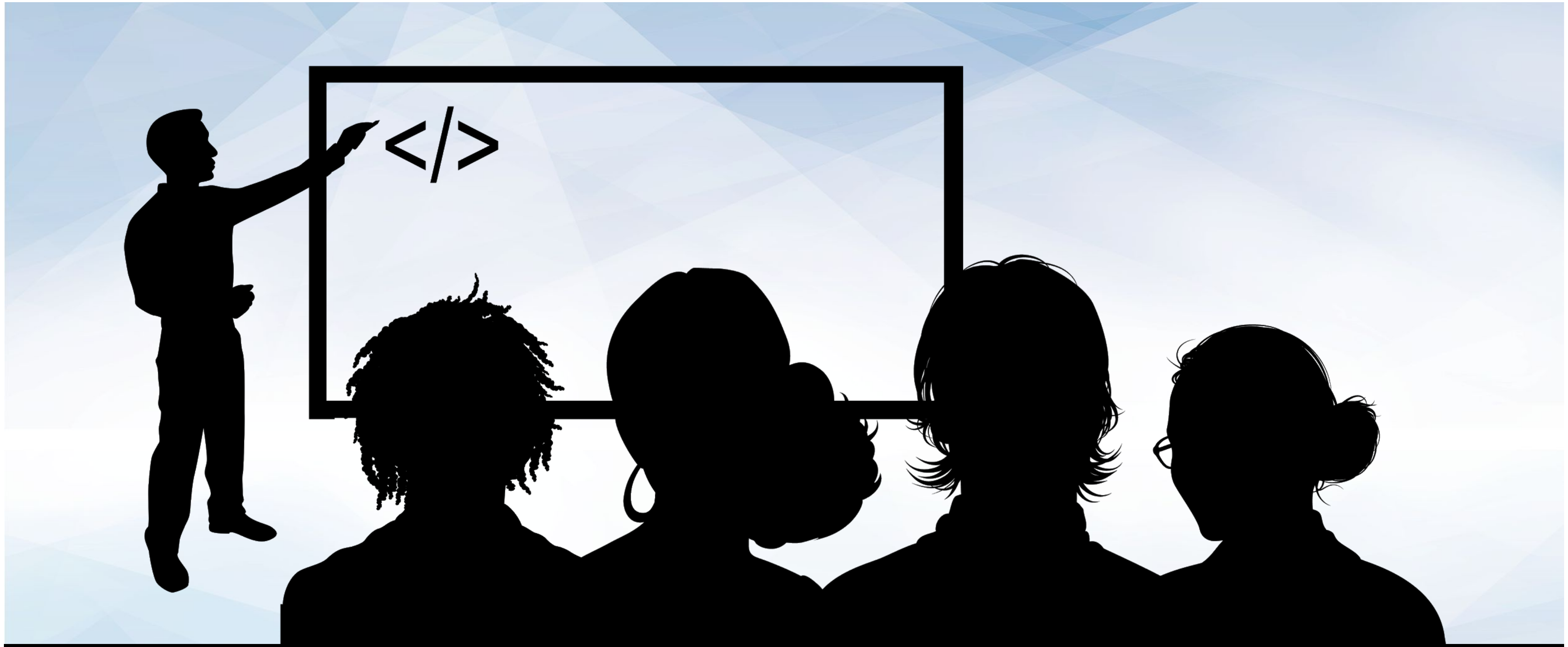
Captain Planet: The Game!



Superpowers: Change Sizes!

Superpowers: Invisibility

Move Controls



Instructor Demonstration

Captain Planet: The Game!

Group Activity:
Pseudocode Captain Planet

Suggested Time:
7 minutes

Group Activity: Pseudocode Captain Planet

Examine the code for the Captain Planet game. Then, describe how this code works in five steps.

1.
2.
3.
4.
5.

Suggested Time: 7 minutes

Pseudocoding Captain Planet

Solution:

01 Create an initial HTML layout using Bootstrap.

02 Add a reference to jQuery.

03 Assign unique class names to key buttons and images.

04 Use jQuery to capture when the corresponding buttons are clicked, using the `$()` identifier with the class name inside.

05 Create code that changes the CSS of target classes in response to click events.

Activity:
Create a Captain Planet Superpower

Suggested Time:
12 minutes

Activity: Create a Captain Planet Superpower

Review the jQuery API documentation (api.jquery.com). Then, add a button of your own that gives Captain Planet a new power.

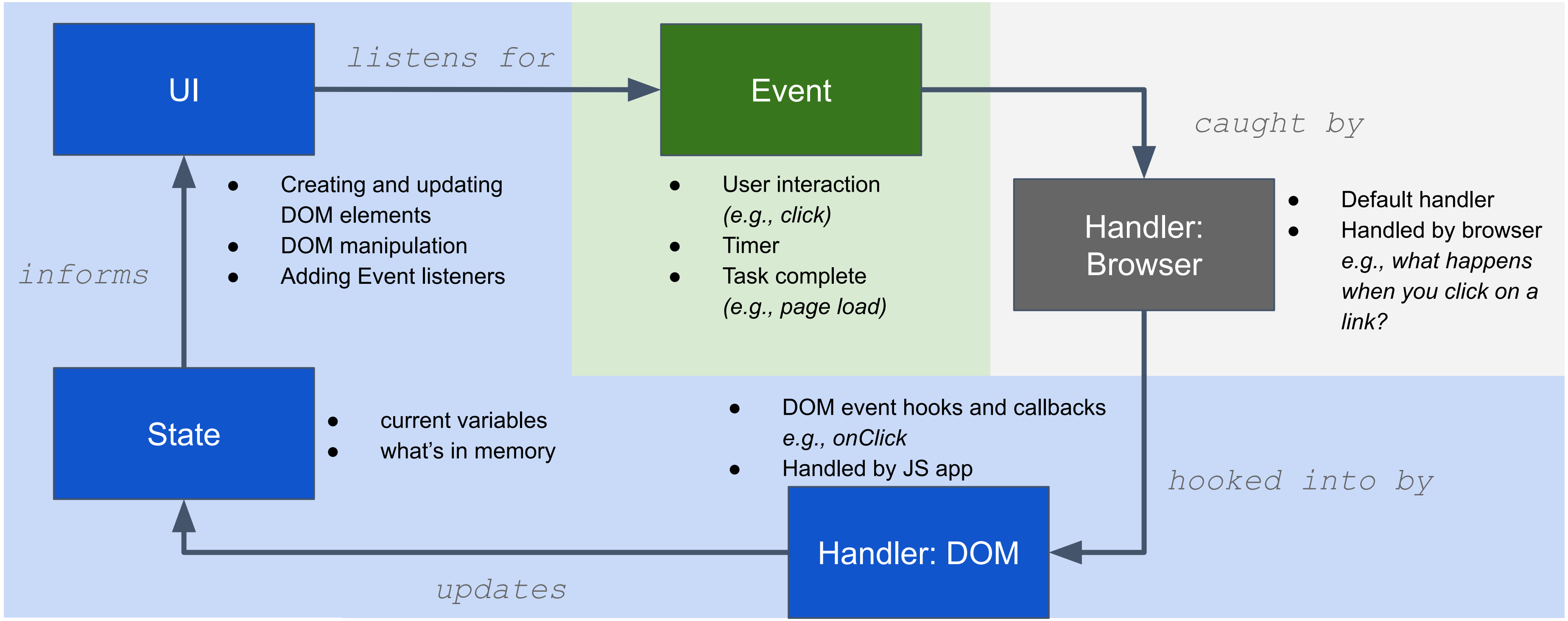
Examples:
Click to…stretch Captain Planet.
Click to…trigger a maniacal laugh.
Click to…create clones of Captain Planet.
Click to…create a shield **(hint: border)**.
Click to…create fire or water **(hint: images)**.

Suggested Time: 12 minutes

DOM Manipulation Recap

Event loop

Handling user interaction



- Creating and updating DOM elements
- DOM manipulation
- Adding Event listeners

- User interaction *(e.g., click)*
- Timer
- Task complete *(e.g., page load)*

- Default handler
- Handled by browser *e.g., what happens when you click on a link?*

- current variables
- what's in memory

- DOM event hooks and callbacks *e.g., onClick*
- Handled by JS app

What is “UI”?

“UI” === “User Interface”

The User Interface is basically what the user sees and interacts with

- Things to read (e.g., text)
- Things to see (images, colors, videos)
- Things to hear (audio)
- Things that look clickable
- Things that look scrollable
- Things that look draggable

etc...

All of this is created with HTML and CSS

Without Events and Event Handlers, **UI is just like paint on a wall**

What is an “Event”?

“Event” === “Something has happened”

The browser has “sensed” something has happened

Some of the things that can happen

- User clicked something
- User rolled over something
- A timer has gone off
- The document has finished loading
- User scrolled
- A request to a remote API has responded

etc...

All of this is noticed by the browser

What do I mean by “Handler”?

“Handler” === “Our event triggered code”

What we want to happen after an event

The browser gives us “hooks” to the events it senses:

- `click, mousedown, mouseup`
- `mouseover, mouseenter, mousemove, mouseleave`
- `setInterval, setTimeout`
- `DOMContentLoaded`
- `scroll`
- `fetch('http://example.com/movies.json').then(response => response.json())`

etc...

All of this is written by us as Javascript

jQuery DOM manipulation

We use the jQuery $( ) identifier to capture HTML elements:

`$(“.classname”)`	`$(“div”)`
`$(“#idname”)`	`$(“p”)`

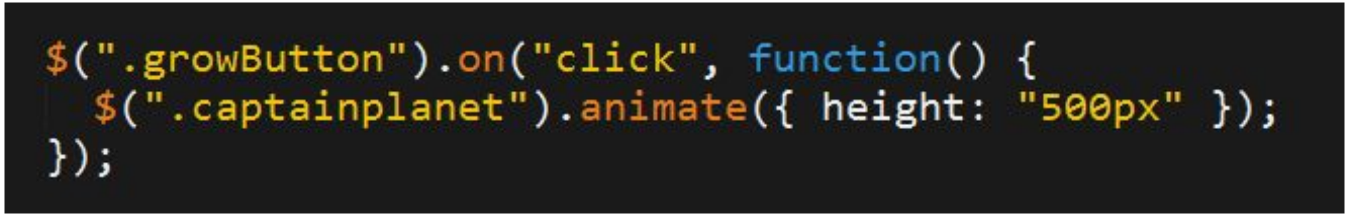
Then, we tie the element to a jQuery method of our choice to capture events:

`.on(“click”)`	`.ready()`

Finally, we modify the selected element or add or remove elements from the DOM:

`.animate()`	`.append()`	`.remove()`

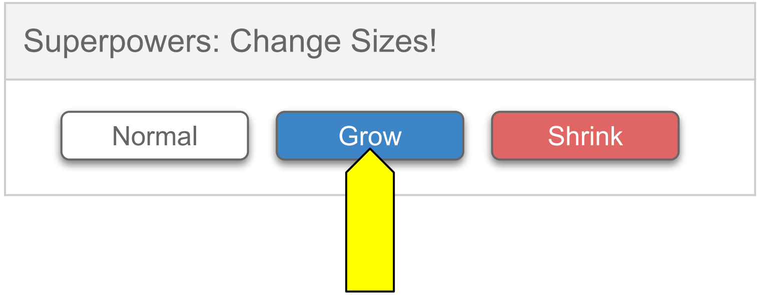
jQuery: A Common Example



```
$(".growButton").on("click", function() {
    $(".captainplanet").animate({ height: "500px" });
});
```

01 Click the Grow button.

02 Make Captain Planet grow.



Superpowers: Change Sizes!

Use Documentation When Needed: api.jquery.com

Group Challenge:
Fridge Game

Suggested Time:
35 minutes

Group Challenge: Fridge Game

Working in your groups, complete the code for the fridge game such that:

JavaScript dynamically generates buttons for each of the letters on the screen.

Clicking any of the buttons causes the same letter to be displayed on the screen.

Clicking the Clear button erases all of the letters from the fridge.

Note: This is a challenging activity. You may want one person in the group to type the code while the other two watch to catch bugs and research code snippets when necessary.

Suggested Time: 35 minutes

This next section is **heavy** on theory.

Scope

Scope: boundaries for variables

Variables have accessibility

Example: Roommates

House rules:

1. Each bedroom is private
   e.g., Hank can't use what's in Bill's room

2. Living room is a shared space
   e.g., Everyone can use the couch

Hank's Room
Penny's Room
Bill's Room
Living Room

Function Scope: boundaries for variables

Variables have accessibility

Example: Roommate = Function

Scope rules:

1. Each function has a **local scope**
   Variables declared in a function are only accessible in that function.

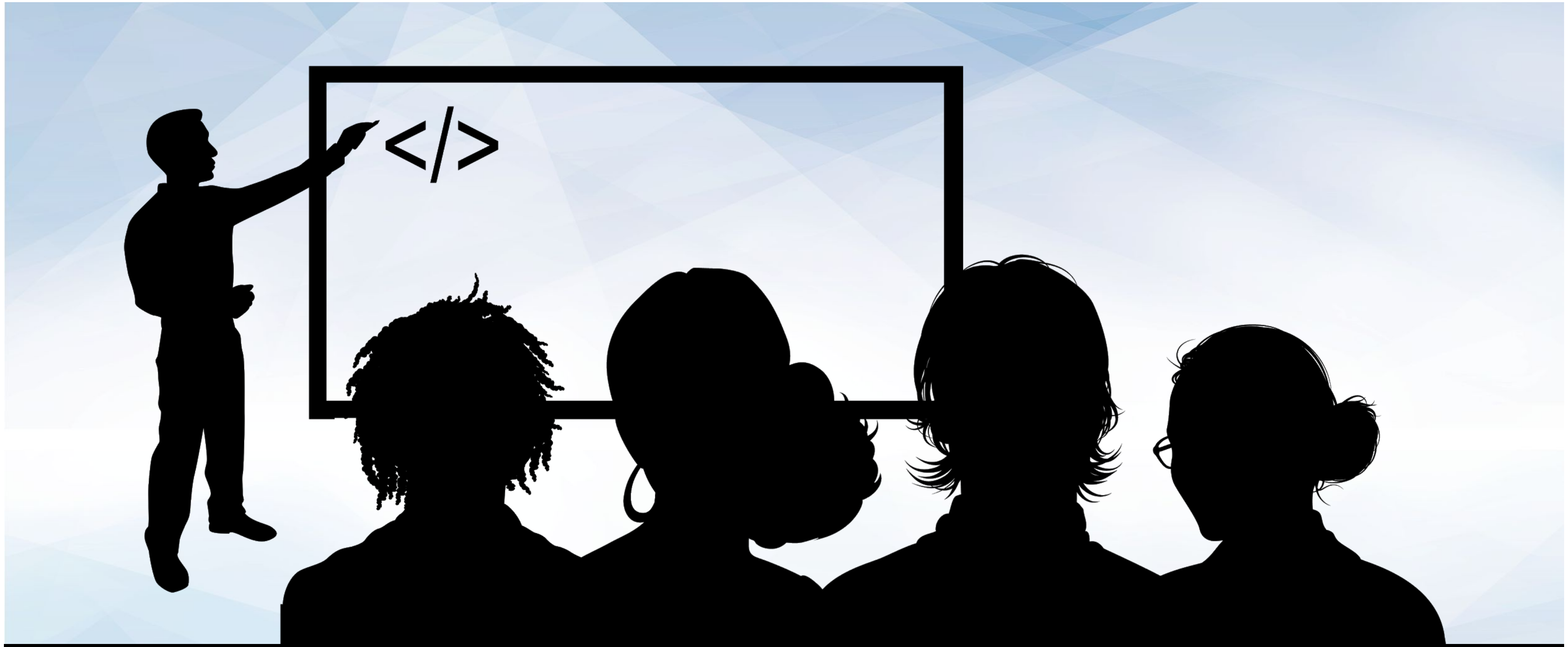
2. Outside all the functions is **global space**
   Variables declared outside all functions are accessible to all functions

```
// Declared in global space, visible to all functions
var userName = “John Young”;

function printCount() {
  // local space
  var numLines = 5; // only visible in printCount
  for(var i = 0; i < numLines; i++) {
    console.log(counter);
  }
}

function checkUserName(validName) {
  // local space
  // validName is only visible in checkUserName
  if(userName === validName) {
    console.log(“This is a valid name”);
    return true;
  }

  return false;
}
```

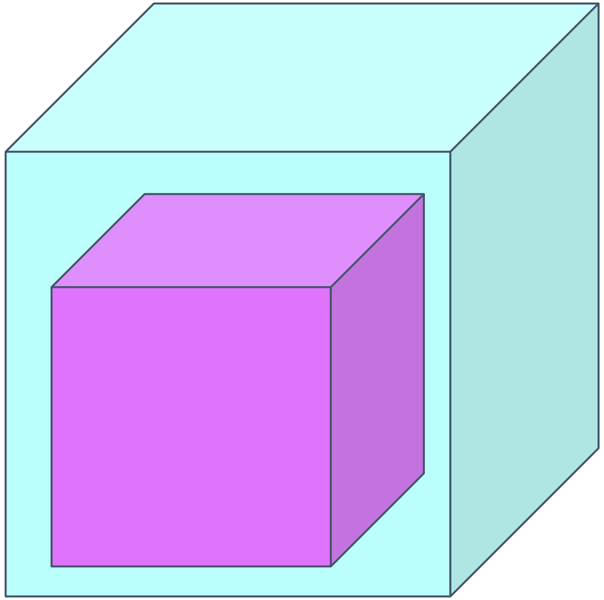


Instructor Demonstration

Function Scope

Scope = Boxes in Boxes

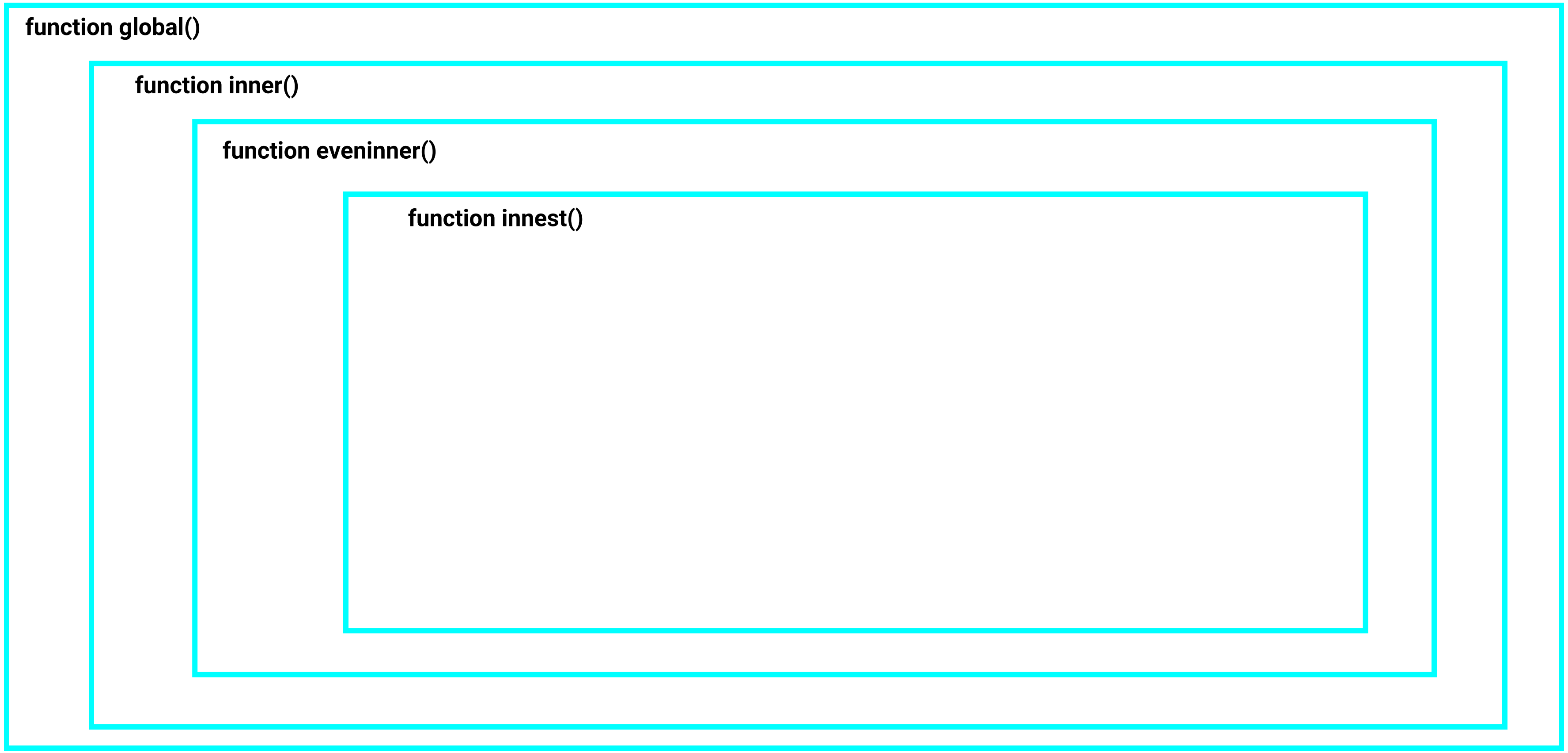
Scope impacts which variables can be accessed by which function.



Lexical Scope

- Lexical scope is the **author-time scope** as defined by blocks of code
- In Javascript, curly **brackets { }** indicate blocks of code.
- Scope is not defined at runtime, rather it is **accessed** at runtime.

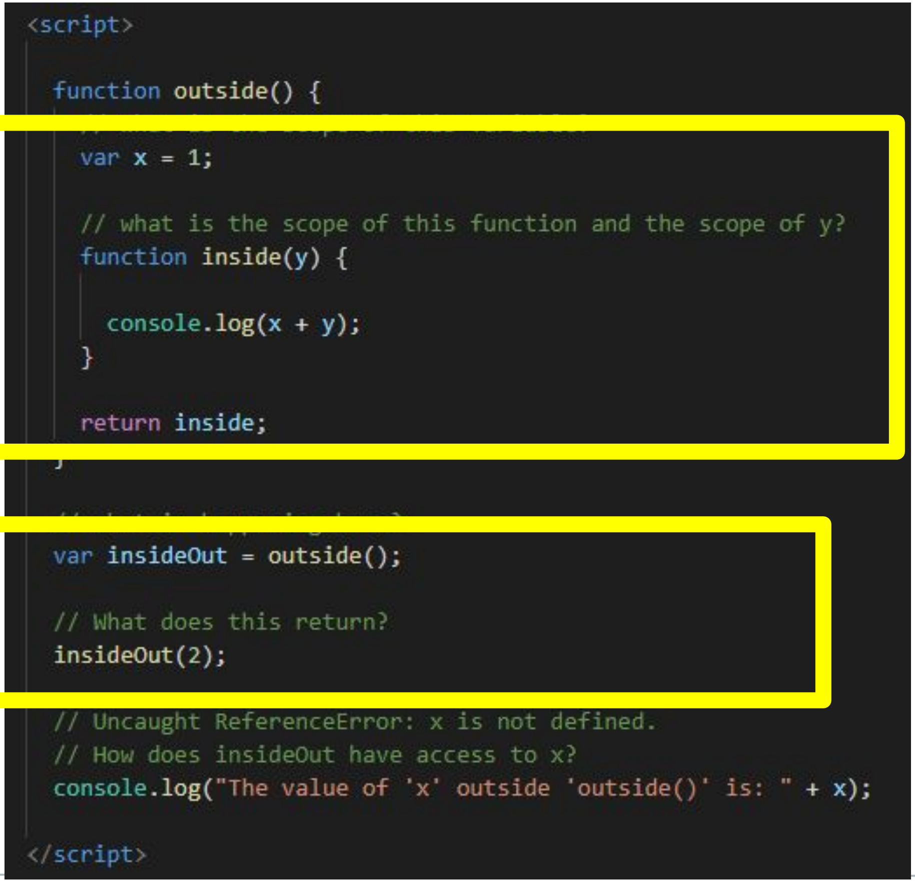
Scope = Boxes in Boxes



JavaScript Lexical Scope Example

Here, **inside** is clearly able to access the variables of its **parent function, outside**.

How does **insideOut** have access to **x**?



```
<script>

  function outside() {
    [illegible]
    var x = 1;

    // what is the scope of this function and the scope of y?
    function inside(y) {

      console.log(x + y);
    }

    return inside;
  }

  [illegible]
  var insideOut = outside();

  // What does this return?
  insideOut(2);

  // Uncaught ReferenceError: x is not defined.
  // How does insideOut have access to x?
  console.log("The value of 'x' outside 'outside()' is: " + x);

</script>
```

Activity:
Lexical Scope 1

Suggested Time:
10 minutes

Activity: Lexical Scope 1

Review the file sent to you and explain the following to the person sitting next to you:

- What do the terms *parent function* and *child function* mean?
- Why can child functions access parent variables, but not vice versa?

Be prepared to share your answers!

Suggested Time: 10 minutes

Activity:
Lexical Scope 2

Suggested Time:
7 minutes

Activity: Lexical Scope 2

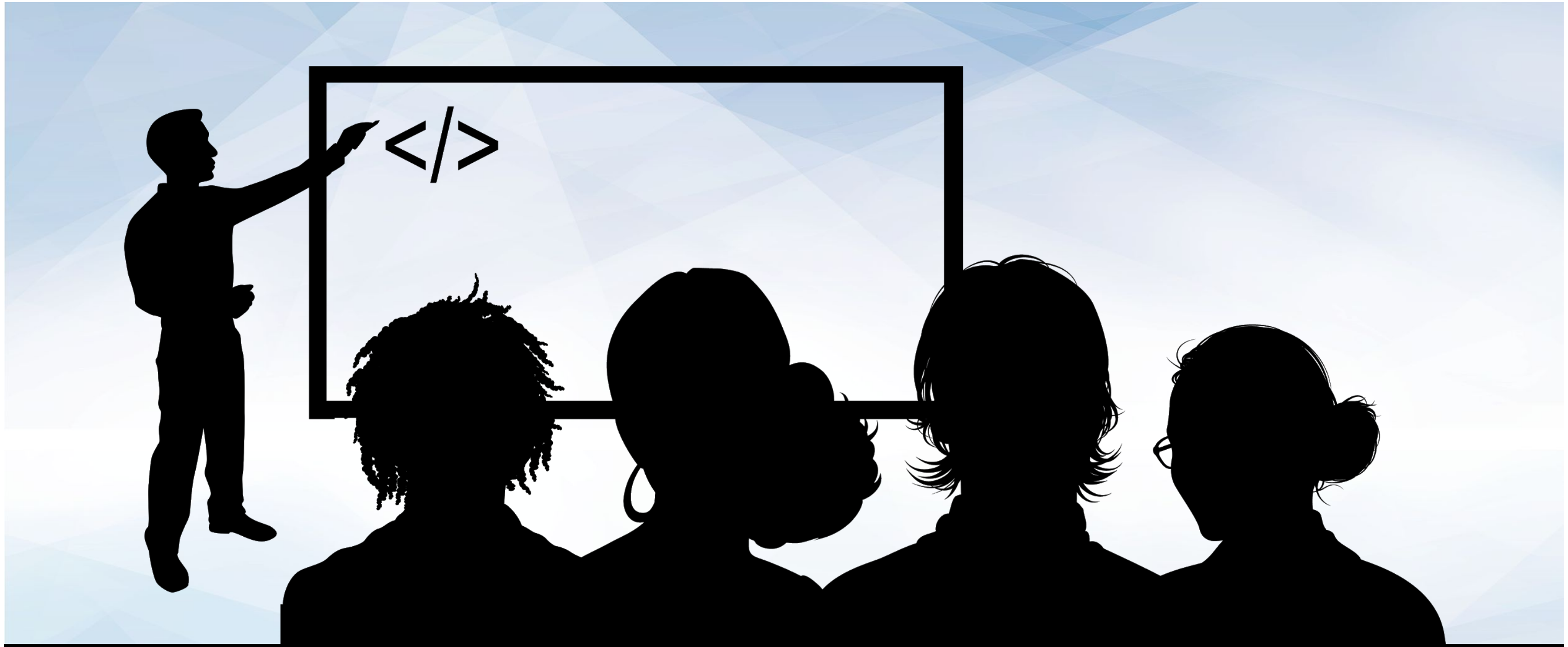
Take a few moments to dissect the code just sent to you.

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*.

Suggested Time: 7 minutes



Instructor Demonstration

Lexical Scope 2

Activity:
Lexical Scope 3

Suggested Time:
7 minutes

Activity: Lexical Scope 3

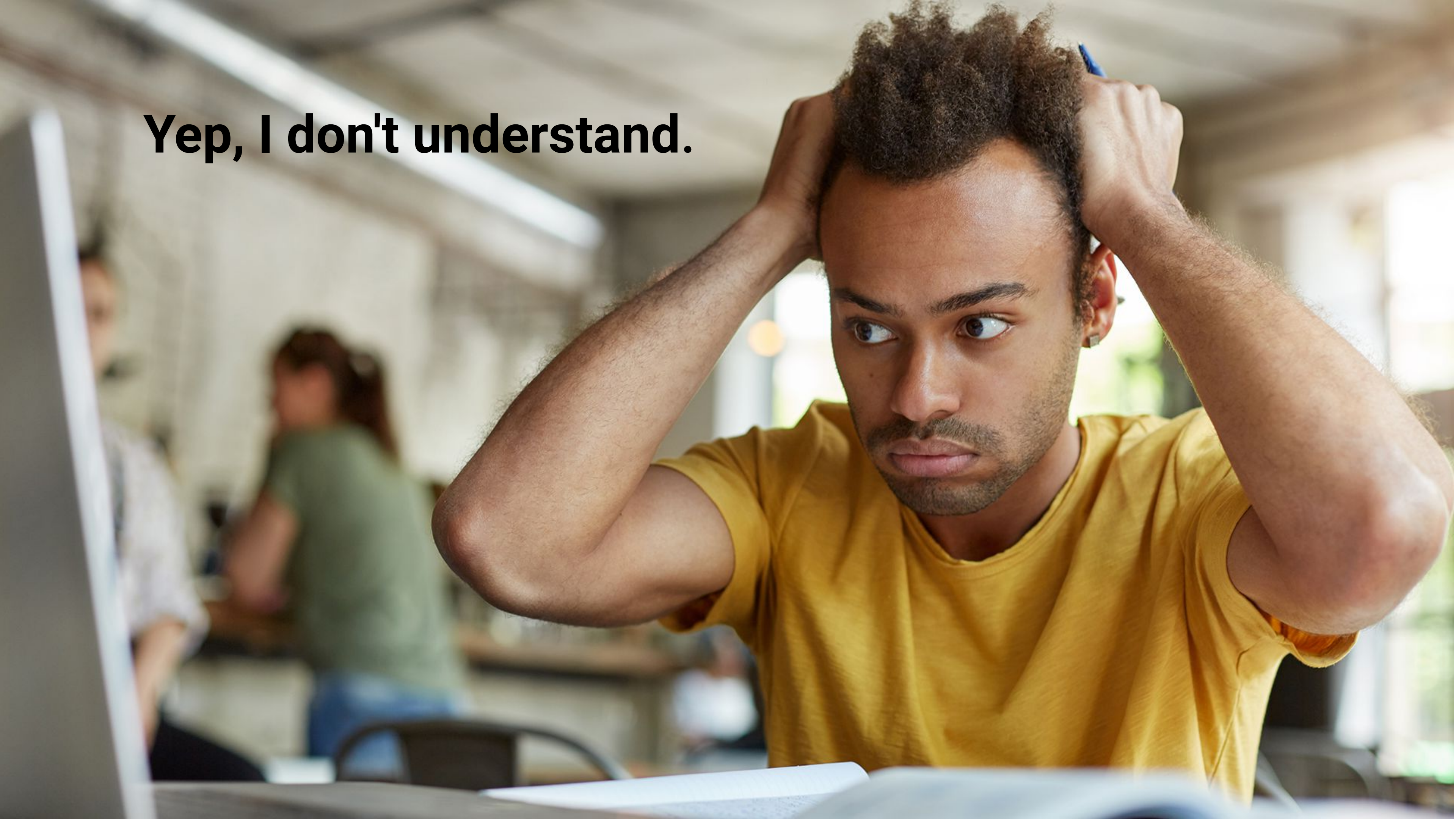
Take a few moments to dissect the code just sent to you.

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*.

Suggested Time: 7 minutes



Yep, I don't understand.

If you'd like to learn more, here's a helpful article:

What You Should Already Know about JavaScript Scope

[spin.atomicobject.com]

Understanding Scope in JavaScript

[scotch.io]

Challenge:

Color Corrector:
Build a Brain Teaser

Suggested Time:
20 minutes *plus* additional 20 minutes at home

Color Corrector: Build a Brain Teaser

Choose the color of the word shown from the list below:

teal

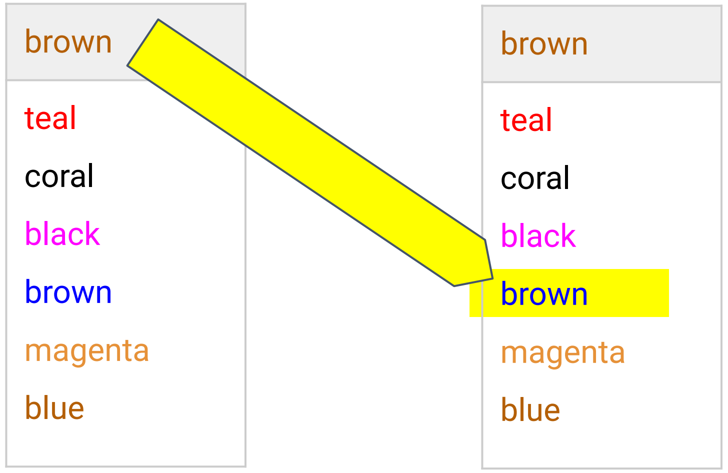
brown
magenta
blue
teal
coral
black

Challenge: Color Corrector: Build a Brain Teaser

Using the files sent to you as a starting point, add the missing code so that the Color Corrector game works correctly.

To win, choose the word that matches the color of the text at the top of the column.

Example:



Suggested Time: 20 minutes

Questions?