GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `svodnik/JS-SF-8-resources` repo to your computer:
   - Open the terminal
   - `cd` to the `JSD/JS-SF-8-resources` directory
   - Type **`git pull`** and press **return**
2. In your code editor, open the following folder: `JSD/JS-SF-8-resources/04-functions-scope/starter-code`

JAVASCRIPT DEVELOPMENT

FUNCTIONS AND SCOPE

LEARNING OBJECTIVES

At the end of this class, you will be able to

- Describe how parameters and arguments relate to functions
- Create and call a function that accepts parameters to solve a problem
- Define and call functions defined in terms of other functions
- Return a value from a function using the `return` keyword
- Define and call functions with argument-dependent return values
- Determine the scope of local and global variables
- Create a program that hoists variables

AGENDA

- Functions
- Variable scope
- The `var`, `let`, and `const` keywords
- Hoisting

WEEKLY OVERVIEW

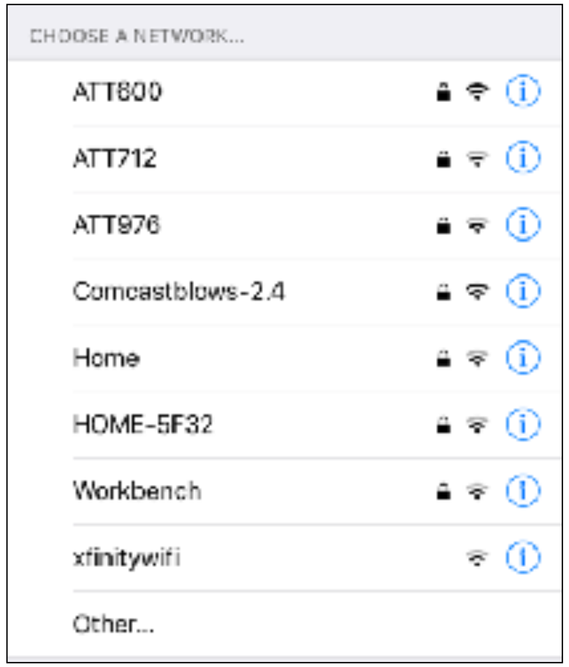
WEEK 3	Loops & Conditionals / Functions & Scope
WEEK 4	Slackbot Lab / Objects & JSON
WEEK 5	Intro to the DOM / Intro to jQuery

EXIT TICKET QUESTIONS

1. Still interested in how git is used in practice, or how ~20 people would collaborate instead of just 2.
2. Why `for` over `while` over `do/while`?
3. Not entirely clear on why the initial 'clone' of the homework was incorrect to do, and how forking from it fixed it.
4. more truthiness and falseyness, please.
5. Using map and forEach as a function to call is still not intuitive to me. I wish I had better understanding of how I'd use it.

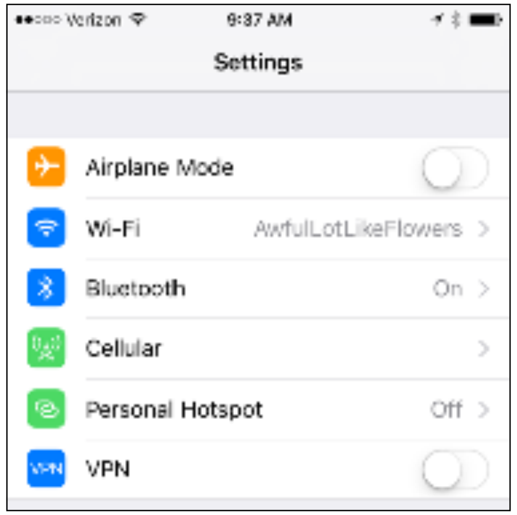
Why do we use different networks to connect to the Internet when we're in different places?

- home
- GA
- in a car
- on BART/MUNI



CHOOSE A NETWORK...

- ATT800
- ATT712
- ATT976
- Comcastblows-2.4
- Home
- HOME-5F32
- Workbench
- xfinitywifi
- Other...



Verizon 9:37 AM

Settings

Airplane Mode	
Wi-Fi	AwfulLotLikeFlowers
Bluetooth	On
Cellular	
Personal Hotspot	Off
VPN	

LAB — CONDITIONALS

EXERCISE

TYPE OF EXERCISE

- Pair

LOCATION

- `starter-code > 0-ages-lab`

TIMING

20 min

1. Write a program that outputs results based on users' age. Use the list of conditions in the `app.js` file.
2. BONUS 1: Rewrite your code to allow a user to enter an age value, rather than hard-coding it into your program. (Hint: Read up on the window.prompt method.)
3. BONUS 3: Rewrite your code to use a switch statement rather than if and else statements.

FUNCTIONS

FUNCTIONS

GROUP STEPS

Allow us to group a series of statements together to perform a specific task

REUSABLE

We can use the same function multiple times

STORE STEPS

Not always executed when a page loads. Provide us with a way to 'store' the steps needed to achieve a task.

FUNCTIONS AND SCOPE

DRY =
DON'T
REPEAT
YOURSELF

FUNCTION DECLARATION SYNTAX

```
function name(parameters) {
  // do something
}
```

FUNCTION DECLARATION EXAMPLE

```
function speak() {
  console.log(“Hello!”);
}
```

FUNCTION EXPRESSION SYNTAX

```
let name = function(parameters) {
  // do something
};
```

FUNCTION EXPRESSION EXAMPLE

```
let speak = function() {
  console.log(“Hello!”);
};
```

ARROW FUNCTION SYNTAX

```
let name = (parameters) => {
  // do something
};
```

ARROW FUNCTION EXAMPLE

```
let speak = () => {
  console.log(“Hello!”);
};
```

EXERCISE — WRITING FUNCTIONS

EXERCISE

KEY OBJECTIVE

- Practice defining and executing functions

TYPE OF EXERCISE

- Individual/paired

LOCATION

- `starter-code > 0-functions-exercise` (part 1)

EXECUTION

4 min	1. Follow the instructions under Part 1

CALLING A FUNCTION

```
function pickADescriptiveName() {
    // do something
}
```

To run the function, we need to *call* it. We can do so like this:

```
pickADescriptiveName();
```

Function name + parentheses

FUNCTION EXPRESSION VS FUNCTION DECLARATION

- Function expressions define functions that can be used anywhere in the scope where they're defined.
- You can call a function that is defined using a function declaration before the part of the code where you actually define it.
- Function expressions must be defined before they are called.

OBJECT METHODS ARE FUNCTIONS

properties

method

value is a function

calling a method

```
let person = {
  fName: 'Kamala',
  lName: 'Harris',
  speak: function() {
    console.log("Hello world!");
  }
}

person.speak()
=> "Hello world!"
```

PARAMETERS

DOES THIS CODE SCALE?

```
function helloVal () {
  console.log('hello, Val');
}

function helloOtto () {
  console.log('hello, Otto')
}
```

USING A PARAMETER

parameter

```
function sayHello(name) {
  console.log('Hello ' + name);
}
```

argument

```
sayHello('Val');
=> 'Hello Val'

sayHello('Otto');
=> 'Hello Otto’
```

USING MULTIPLE PARAMETERS

multiple parameter names separated by commas

```
function sum(x, y, z) {
  console.log(x + y + z)
}

sum(1, 2, 3);
=> 6
```

USING DEFAULT PARAMETERS

default value to set for parameter if no argument is passed when the function is called

```
function multiply(x, y = 2) {
  console.log(x * y)
}

multiply(5, 6);
=> 30 // result of 5 * 6 (both arguments)
multiply(4);
=> 8 // 4 (argument) * 2 (default value)
```

EXERCISE — READING FUNCTIONS

EXERCISE

KEY OBJECTIVE

- Given a function and a set of arguments, predict the output of a function

TYPE OF EXERCISE

- Groups of 2 - 3

LOCATION

- `starter-code > 0-functions-exercise` (part 2)

EXECUTION

3 min

1. Look at Part 2 A and B. Predict what will happen when each function is called.

EXERCISE — READING FUNCTIONS

EXERCISE

KEY OBJECTIVE

- Create and call a function that accepts parameters to solve a problem

TYPE OF EXERCISE

- Groups of 2 - 3

LOCATION

- `starter-code > 0-functions-exercise` (part 3)

EXECUTION

8 min

1. See if you can write one function that takes some parameters and combines the functionality of the *makeAPizza* and *makeAVeggiePizza* functions.
2. BONUS: Create your own function with parameters. This function could do anything!

EXERCISE — FUNCTIONS

EXERCISE

KEY OBJECTIVE

- Describe how parameters and arguments relate to functions

TYPE OF EXERCISE

- Turn and Talk

EXECUTION

1 min

1. Summarize why we would use functions in our programs. What purpose do they serve?
2. What is a parameter? What is an argument? How are parameters and arguments useful?

THE `return` STATEMENT

`return` STATEMENT

- Ends function’s execution
- Returns a value — the result of running the function

`return` STOPS A FUNCTION'S EXECUTION

```
function speak(words) {
  return words;

  // The following statements will not run:
  let x = 1;
  let y = 2;
  console.log(x + y);
}
```

`console.log()` vs `return`

`console.log()`

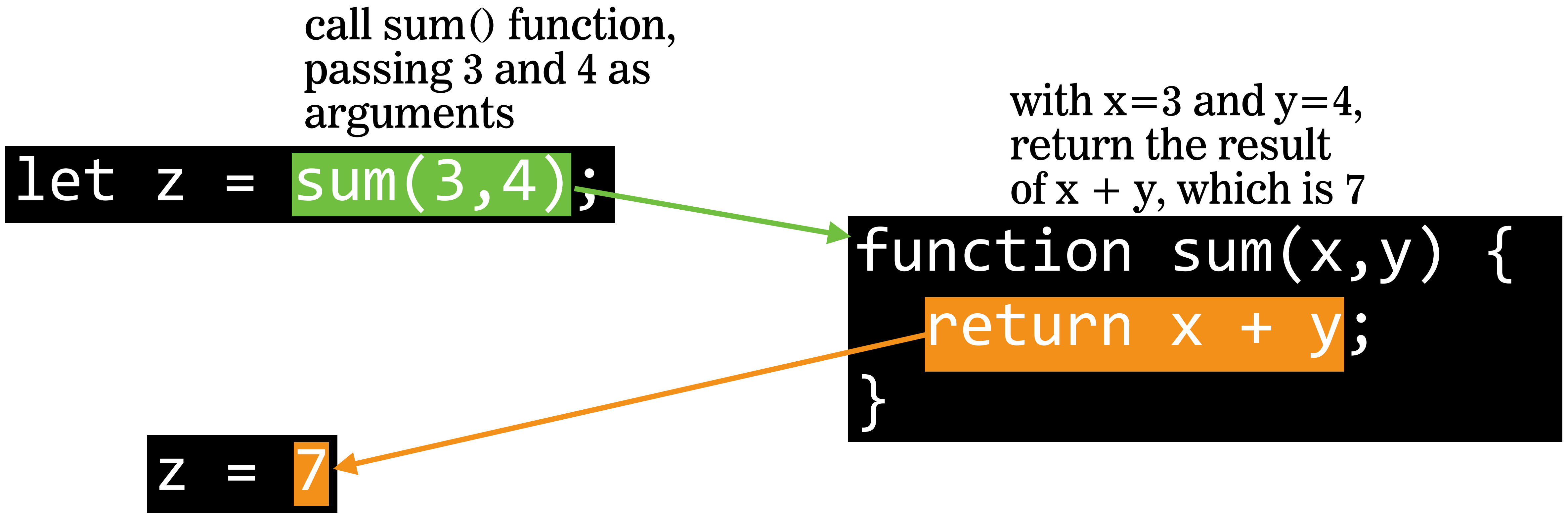
VS

`return`

- Write a value at any point in a program to the browser console
- Helpful for developer in debugging
- Not seen by user or used by app

- Sends a value back wherever the current statement was triggered
- Can use a function to get a value and then use that value elsewhere in your app
- Does not appear in the console unless you're executing commands there

return in action



call sum() function, passing 3 and 4 as arguments

```
let z = sum(3,4);
```

with x=3 and y=4, return the result of x + y, which is 7

```
function sum(x,y) {
  return x + y;
}
```

```
z = 7
```

EXERCISE — FUNCTIONS LAB

EXERCISE

KEY OBJECTIVE

- Create and call a function that accepts parameters to solve a problem

TYPE OF EXERCISE

- Individual or pair

LOCATION

- `starter-code > 1-functions-lab`

EXECUTION

15 min

1. Write code to to calculate a customer's total cost in dollars based on product price, tax rate, shipping cost, and the currency they're using for the purchase (dollars or euros).
2. BONUS: Convert your function to assume a currency of "dollar" by default.

SCOPE

SCOPE

- Describes the set of variables you have access to

GLOBAL SCOPE

- A variable declared outside of a function is accessible everywhere, even within functions. Such a variable is said to have **global scope**.

a variable declared outside of the function is in the global scope

```
let temp = 75;
function predict() {
  console.log(temp); // 75
}
console.log(temp); // 75
```

LOCAL SCOPE

- A variable declared within a function is not accessible outside of that function. Such a variable is said to have **local scope**.

```
let temp = 75;
function predict() {
  let forecast = ‘Sun’;
  console.log(temp + " and " + forecast); // 75 and Sun
}
console.log(temp + " and " + forecast);
// ‘forecast’ is undefined
```

a variable declared within a function is in the local scope of that function

a local variable is not accessible outside of its local scope

BLOCK SCOPE

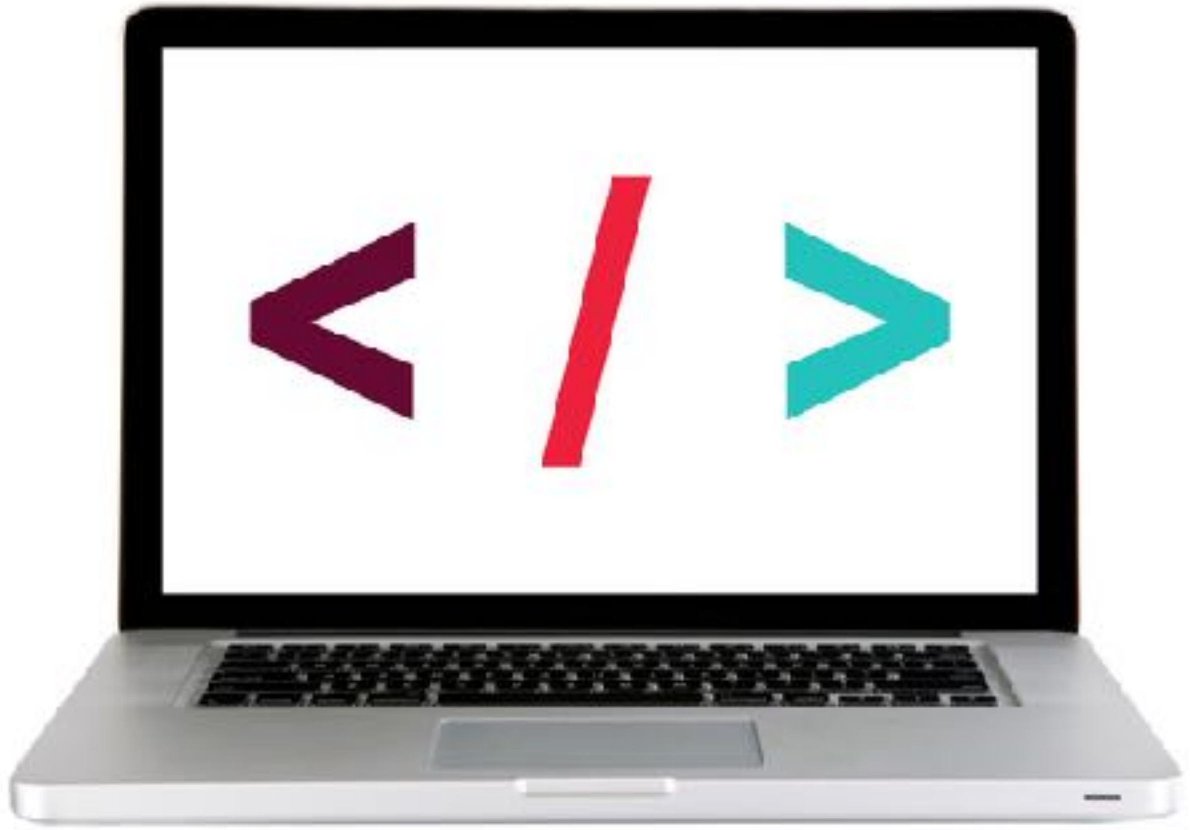
- A variable created with `let` or `const` creates local scope within any block, including blocks that are part of loops and conditionals.
- This is known as **block scope**.

`let` creates a local variable within any block, such as an `if` statement

```
let temp = 75;
if (temp > 70) {
  let forecast = ‘It’s gonna be warm!’;
  console.log(temp + “! ” + forecast); // 75! It’s gonna be warm!
}
console.log(temp + “! " + forecast); // ‘forecast’ is undefined
```

a variable with block scope is not accessible outside of its block

LET'S TAKE A CLOSER LOOK



EXERCISE — SCOPE

EXERCISE

KEY OBJECTIVE

- Determine the scope of local and global variables

TYPE OF EXERCISE

- Turn and Talk

EXECUTION

3 min

1. Describe the difference between global and local scope
2. Collaborate to write code that includes at least one variable with local scope and one variable with global scope

LAB — SCOPE

KEY OBJECTIVE

- Determine the scope of local and global variables

TYPE OF EXERCISE

- Pairs

LOCATION

- `starter code > 1-scope-lab`

EXECUTION

5 min

1. Open the index.html file in your browser, view the console, and examine the error.
2. Follow the instructions in `js > main.js` to complete parts A and B.

`var`, `let`, `const`, AND SCOPE

- `var` obeys the scoping rules we've just seen
  - "generic" way to create variables
- `let` and `const` are newer keywords with different scoping rules
  - local scope within functions **and** within any block (including loops and conditionals)

let

- used in the same situations as `var`, but with different scoping rules for code blocks

```
let results = [0,5,2];
```

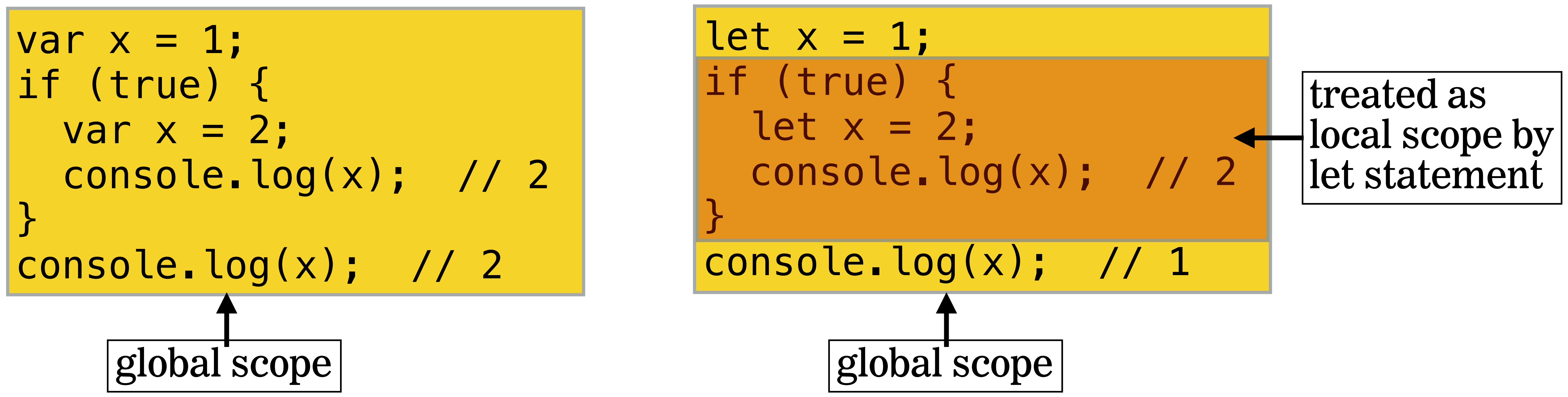
`const`

- used to declare constants
    - **immutable**: once you've declared a value using `const`, you can't change the value in that scope
    - by contrast, variables declared with `var` or `let` are **mutable**, meaning their values can be changed
- by convention, constant names use all capital letters

```
const SALESTAX = 0.0875;
```

`let/const vs var`

- `let` & `const` create local scope within any block (including loops and conditionals) but `var` does not



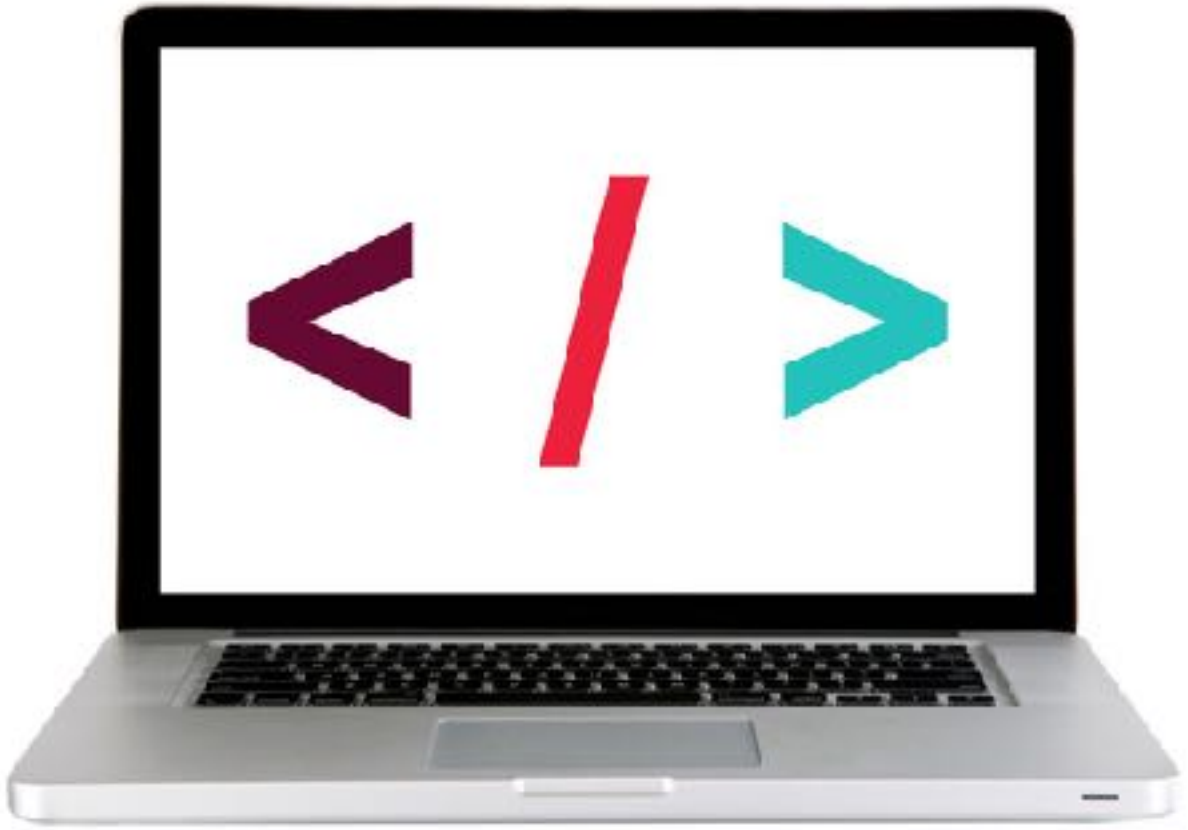
```
var x = 1;
if (true) {
  var x = 2;
  console.log(x);  // 2
}
console.log(x);  // 2
```

```
let x = 1;
if (true) {
  let x = 2;
  console.log(x);  // 2
}
console.log(x);  // 1
```

`var`, `let`, `const`, AND BROWSER SUPPORT

- `let` and `const` are not supported by older browsers
  - see caniuse.com, search on `let`
- babel.js (babeljs.io) allows you to transpile newer code into code that works with older browsers as well
- `let` and `var` are very commonly used, and we will use a combination of these in class (plus the occasional `const`)

LET'S TAKE A CLOSER LOOK



EXERCISE — VAR, LET, AND CONST

EXERCISE

KEY OBJECTIVE

- Distinguish between `var`, `let`, and `const`

TYPE OF EXERCISE

- Individual or pairs

EXECUTION

2 min

1. Draw the table shown on the whiteboard, which compares a few aspects of var, let, and const usage.
2. Complete the table.

`var`, `let`, AND `const`

keyword	local scope	can you change the value in the current scope?	browser support
`var`	within the code block of a **function** only	yes	all browsers
`let`	within any code block	yes	**only modern browsers**
`const`	within any code block	**no**	**only modern browsers**

LAB — LET, VAR, AND CONST

KEY OBJECTIVE

- Determine the scope of local and global variables

TYPE OF EXERCISE

- Pairs

LOCATION

- `starter code > 4-let-var-const-lab`

EXECUTION

5 min

1. Open the index.html file in your browser, view the console, and examine the error.
2. Follow the instructions in `js > app.js` to complete parts A and B.

HOISTING

- JavaScript’s behavior of moving declarations to the top of a scope.
- This means that you are able to use a function or a variable before it has been declared.
- Variables declared with `var` are hoisted
- Variables declared with `let` and `const` are not hoisted

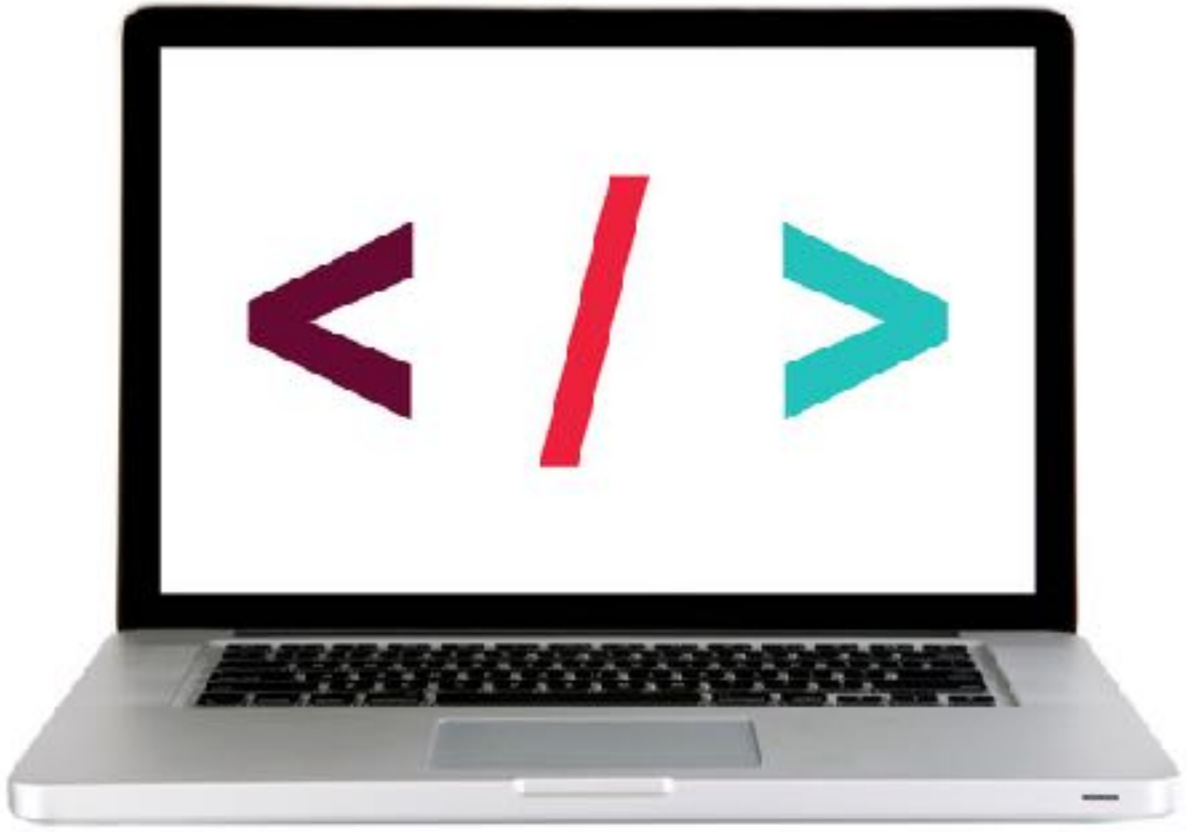
FUNCTIONS AND HOISTING

- Function expressions are treated like other variables
  - when declared with `var`, only the name is hoisted, not the value
  - when declared with `let`, they are not hoisted
- Function declarations are treated differently
  - the code for the entire function is hoisted along with a function declaration

FUNCTIONS AND HOISTING

function type	function name hoisted?	function content hoisted?
function declaration	yes	yes
function expression using `let`	no	no
function expression using var	yes	no

LET'S TAKE A CLOSER LOOK



EXERCISE — HOISTING

EXERCISE

KEY OBJECTIVE

- Create a program that hoists variables

TYPE OF EXERCISE

- Groups of 3

EXECUTION

2 min

1. Examine the code on the whiteboard.
2. Discuss with your group which parts of the code are hoisted.
3. Predict the result of each of the first four statements.

LEARNING OBJECTIVES - REVIEW

- Describe how parameters and arguments relate to functions
- Create and call a function that accepts parameters to solve a problem
- Define and call functions defined in terms of other functions
- Return a value from a function using the `return` keyword
- Define and call functions with argument-dependent return values
- Determine the scope of local and global variables
- Create a program that hoists variables

NEXT CLASS PREVIEW

Hubot Lab

- Install and configure all utilities needed to run a Hubot
- Write scripts that allow your Hubot to interact with users of the class Slack organization

Exit Tickets!

(Class #4)

Q&A