

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the vodnik/JS-SF-14-resources repo to your computer:
 - Open the terminal
 - cd to the ~/Documents/JSD/JS-SF-14-resources directory
 - Type **git pull** and press **return**
2. In your code editor, open the following folder:
Documents/JSD/JS-SF-14-resources/04-scope-objects

JAVASCRIPT DEVELOPMENT

SCOPE & OBJECTS

LEARNING OBJECTIVES

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

- › Determine the scope of local and global variables
- › Describe what hoisting does
- › Identify likely objects, properties, and methods in real-world scenarios
- › Create JavaScript objects using object literal notation

AGENDA

- Variable scope
- The var, let, and const keywords
- Hoisting
- Objects

SCOPE & OBJECTS

WEEKLY OVERVIEW

WEEK 3

Conditionals & Functions / Scope & objects

WEEK 4

Slack Bot Lab / Scope & objects

WEEK 5

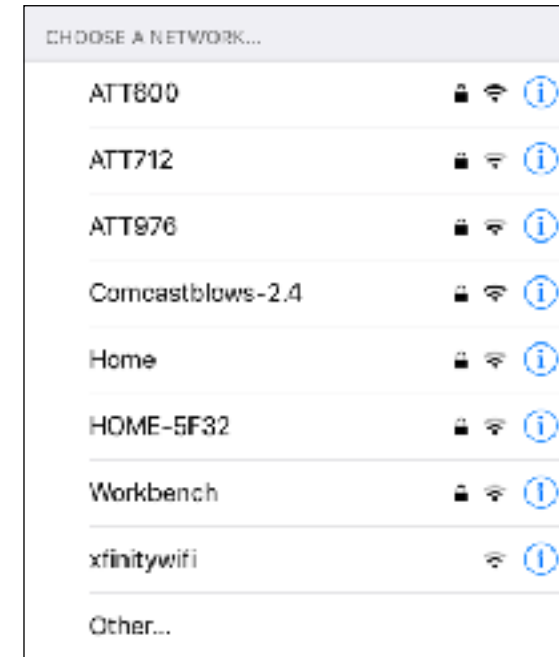
Intro to DOM & jQuery / Events & jQuery

EXIT TICKET QUESTIONS

1. are there any guidelines for choosing which loop works
2. maybe a bit more on the falsey topics

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



SCOPE

GLOBAL SCOPE

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

global variable



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


FUNCTION SCOPE

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

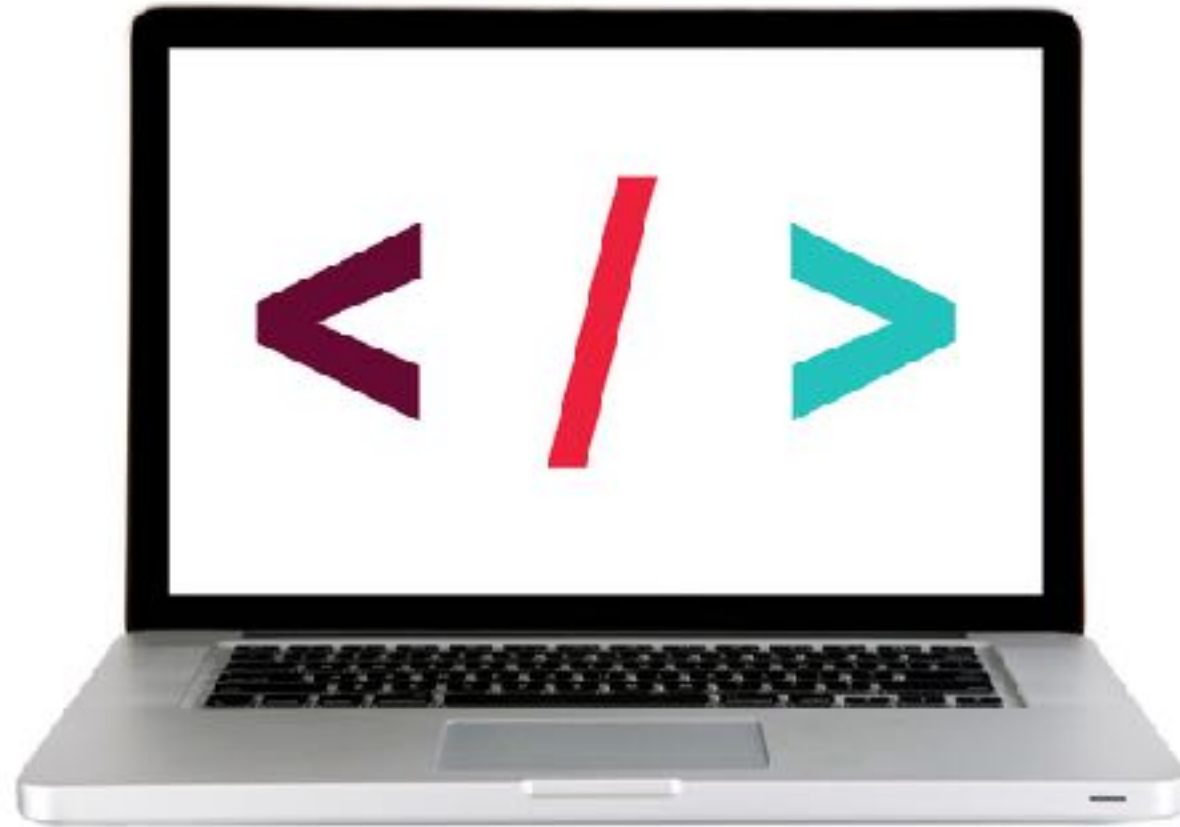
- A variable created with `let` or `const` creates local scope within any block, including blocks that are part of loops and conditionals.
- This is known as **block scope**, which is another type of local scope.

`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 scope, local scope, function scope, and block scope.
2. Collaborate to write code that includes at least
 - one variable with global scope
 - one variable with function scope
 - one variable with block scope.

LAB — SCOPE



KEY OBJECTIVE

- Determine the scope of local and global variables

TYPE OF EXERCISE

- Pairs

LOCATION

- `starter code > 1-scope-lab`

EXECUTION

3 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.

let, const, var, AND SCOPE

let

- let

- » newer keyword (ES6)
- » local scope within functions **and** within any block (including loops and conditionals)

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


const

- const

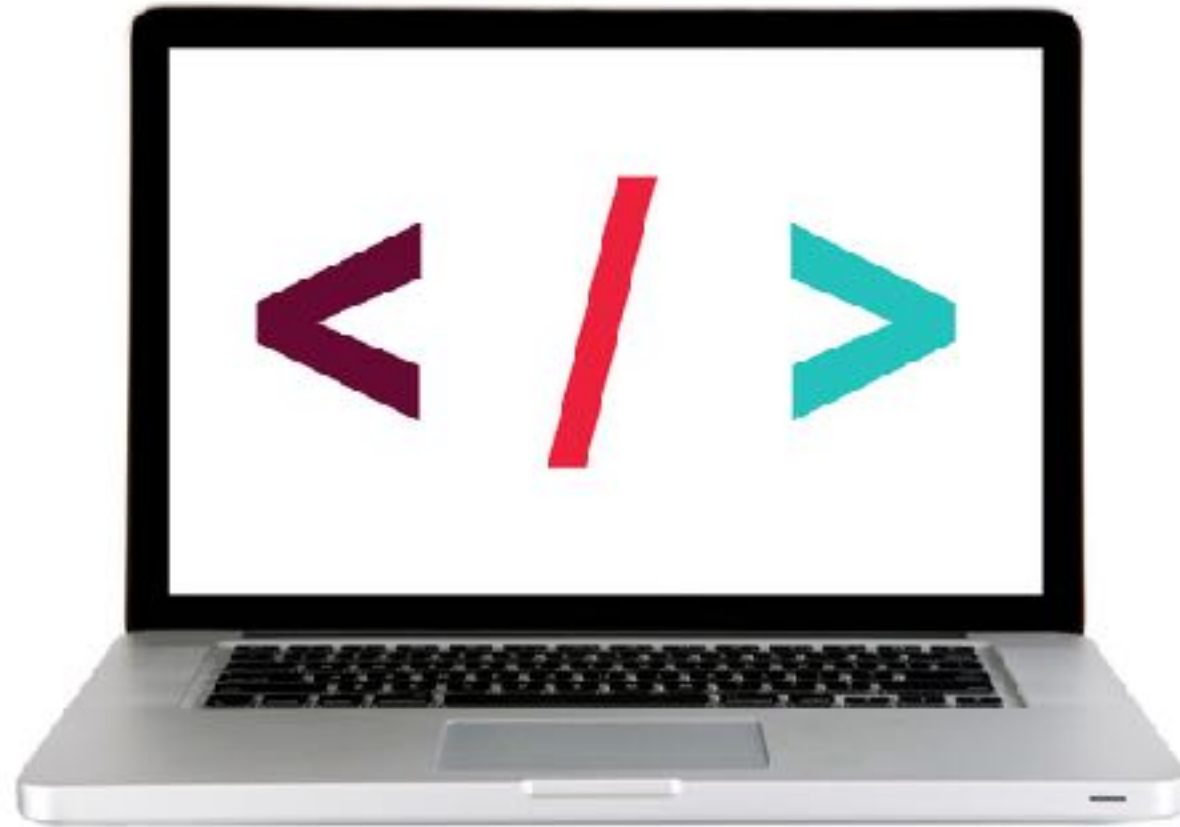
- » newer keyword (ES6)
- » local scope within functions **and** within any block (including loops and conditionals)

- 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

```
const salesTax = 0.0875;
```

LET'S TAKE A CLOSER LOOK



var

- » original JS keyword for creating variables
- » only type of local scope it can create is function scope

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


let/const vs var

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

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

var does not
create local
scope within
a block



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

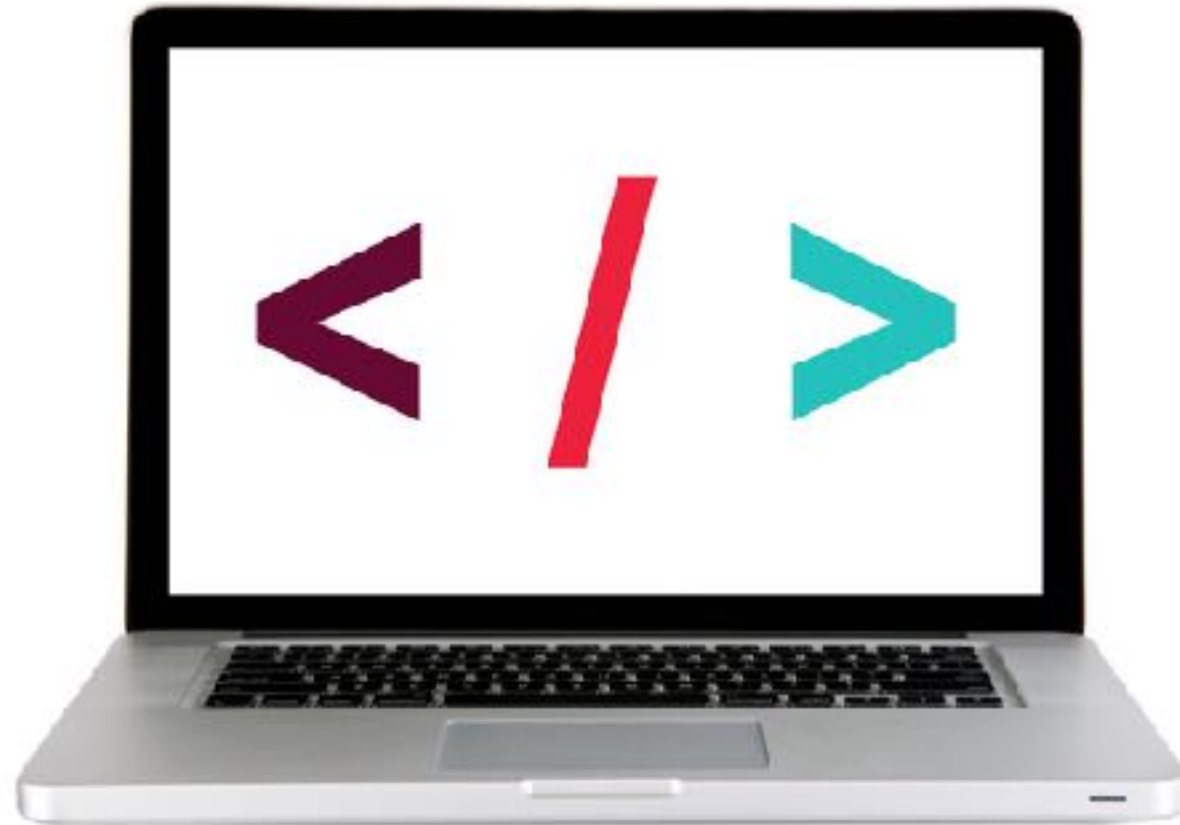

let, const, var, AND BROWSER SUPPORT

- 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
- we will rely on let and const in class

let, const, AND var

keyword	where does it create local scope?	can you change the value in the current scope?	which browsers support it? (modern or all)
let	within any block	yes	only modern browsers
const	within any block	no	only modern browsers
var	within a function block only	yes	all browsers

LET'S TAKE A CLOSER LOOK



LAB — LET, VAR, AND CONST



KEY OBJECTIVE

- Determine the scope of local and global variables

TYPE OF EXERCISE

- Pairs

LOCATION

- `starter code > 2-let-var-const-lab`

EXECUTION

3 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

HOISTING

Variable names declared with `var` are hoisted, but not their values.

Code as written by developer



```
function foo() {  
  console.log("Hello!");  
  var x = 1;  
}
```

Code as interpreted by parser

```
function foo() {  
  var x;  
  console.log("Hello!");  
  x = 1;  
}
```


HOISTING

Variables declared with `let` or `const` are **not** hoisted.

Code as written by developer

```
function foo() {  
  console.log("Hello!");  
  let x = 1;  
}
```

Code as interpreted by parser


```
function foo() {  
  console.log("Hello!");  
  let x = 1;  
}
```


HOISTING

Function declarations are hoisted.

Your code can call a hoisted function before it has been declared

Code as written by developer



```
foo();  
  
function foo() {  
    console.log("Hello!");  
}
```

Code as interpreted by parser

```
function foo() {  
    console.log("Hello!");  
}  
  
foo();
```


HOISTING

Function expressions are treated like other variables

Code as written by developer



```
foo();
```

```
var foo = function() {  
    console.log("Hello!");  
}
```

Code as interpreted by parser

```
var foo;
```

```
foo(); // error: foo is  
      // not a function
```

```
foo = function() {  
    console.log("Hello!");  
}
```


HOISTING

Function expressions are treated like other variables

Code as written by developer

```
foo();  
  
let foo = function() {  
  console.log("Hello!");  
}
```

Code as interpreted by parser

```
foo(); // error: foo is  
      // not defined  
  
let foo = function() {  
  console.log("Hello!");  
}
```

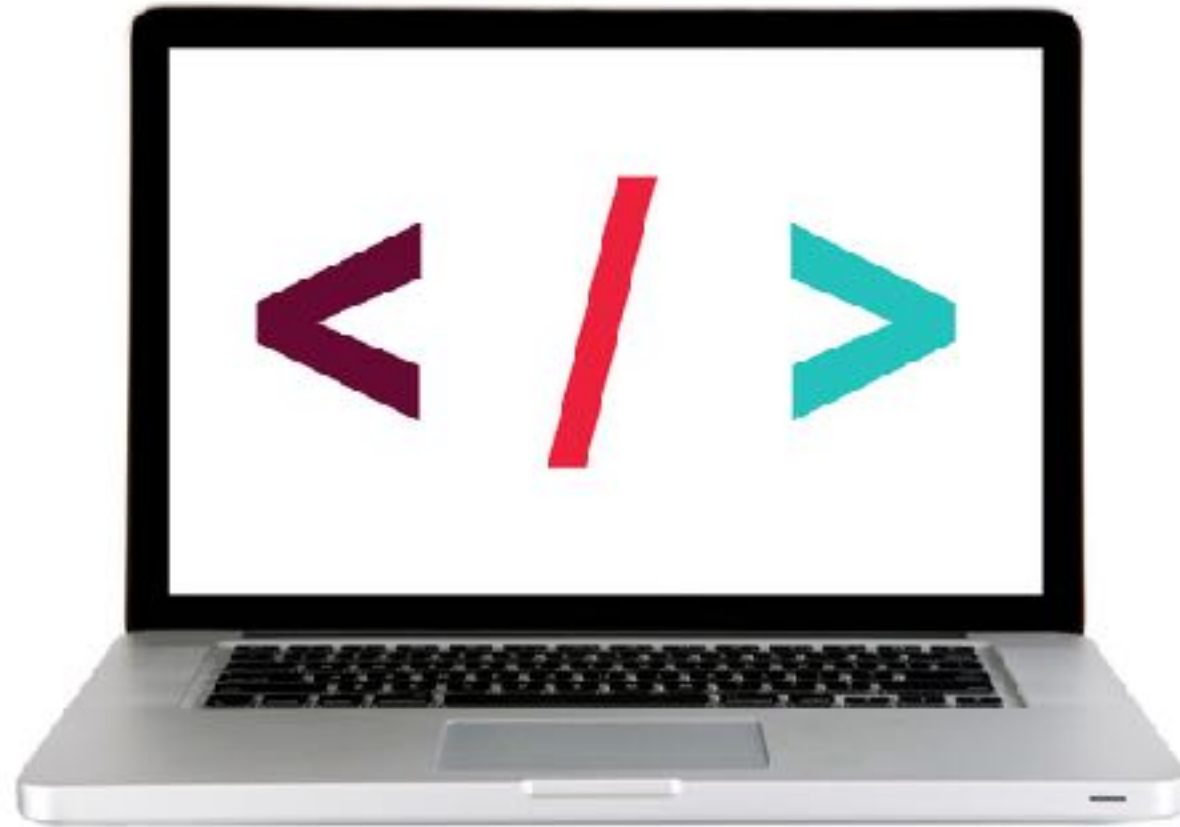

VARIABLES AND HOISTING

keyword	what is hoisted?
let/const	nothing
var	name only

FUNCTIONS AND HOISTING

type	what is hoisted?
declaration	name and content
expression using let/const	nothing
expression using var	name only

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

OBJECTS

EXERCISE — OBJECTS



EXERCISE

KEY OBJECTIVE

- ▶ Create JavaScript objects using object literal notation

TYPE OF EXERCISE

- ▶ Groups of 2-3

TIMING

3 min

1. For the thing you've been assigned, make a list of attributes (descriptions) and actions (things it can do).

OBJECTS ARE A SEPARATE DATA TYPE

STRING

NUMBER

ARRAY

BOOLEAN

OBJECT

AN OBJECT IS A COLLECTION OF PROPERTIES

```
let favorites = {  
  fruit: "apple",  
  vegetable: "carrot",  
}
```

A diagram illustrating the concept of object properties. On the left, the word "properties" is enclosed in an orange box. Two orange arrows originate from the right side of this box. The top arrow points to the text "fruit: 'apple'," which is also enclosed in an orange box. The bottom arrow points to the text "vegetable: 'carrot'," which is also enclosed in an orange box. These two boxes are stacked vertically and are part of a larger code block representing a JavaScript object.

PROPERTY = KEY & VALUE

- A **property** is an association between a key and a value
 - **key**: name (often descriptive) used to reference the data
 - **value**: the data stored in that property



KEY-VALUE PAIR

- A property is sometimes referred to as a **key-value pair**

```
let favorites = {  
  fruit: "apple",  
  vegetable: "carrot"  
}
```

key-value pair

A white rectangular box highlights the line of code 'vegetable: "carrot"' in the code block above. A white arrow points from the text 'key-value pair' to this box.

AN OBJECT IS NOT ORDERED

```
0  [
1   "apple",
2   "pear",
   "banana"
]
```


ARRAY
ordered

```
{
  fruit: "apple",
  vegetable: "carrot",
  fungus: "trumpet mushroom"
}
```

OBJECT
not ordered

A METHOD IS A PROPERTY WHOSE VALUE IS A FUNCTION

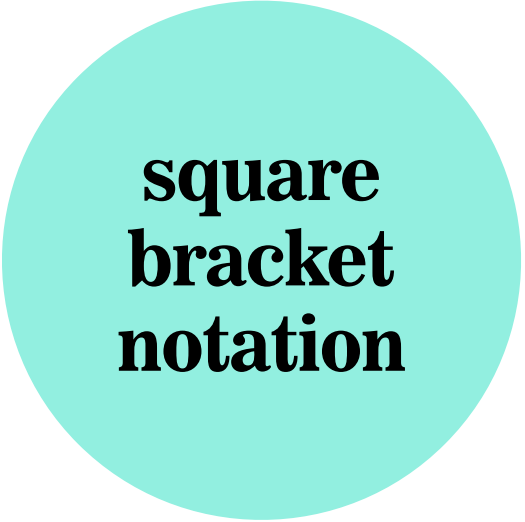
```
let favorites = {  
  fruit: "apple",  
  vegetable: "carrot",  
  declare: function() {  
    console.log("I like fruits and vegetables!");  
  },  
}
```



TWO WAYS TO GET/SET PROPERTIES



dot notation



**square
bracket
notation**

GETTING A PROPERTY VALUE WITH DOT NOTATION

object

object name

getting properties

```
let favorites = {  
  fruit: "apple",  
  veg: "carrot",  
  declare: function() {  
    console.log("I like fruit and veg");  
  }  
}
```

favorites.fruit ← **property name**
> "apple"
favorites.veg
> "carrot"

object name

calling a method

method name

favorites.declare()
> "I like fruit and veg"

SETTING A PROPERTY VALUE WITH DOT NOTATION

object

```
let favorites = {  
  fruit: "apple",  
  veg: "carrot",  
  declare: function() {  
    console.log("I like fruit and veg");  
  }  
}
```

setting properties

```
favorites.fungus = 'shiitake';  
favorites.pet = 'hamster';
```

setting a method

```
favorites.beAmbivalent = function() {  
  console.log("I like other things");  
};
```


GETTING A PROPERTY VALUE WITH SQUARE BRACKET NOTATION

object

object name

getting properties

```
let favorites = {  
  fruit: "apple",  
  veg: "carrot",  
  declare: function() {  
    console.log("I like fruit and veg");  
  }  
}
```

```
favorites[fruit]  
> "apple"  
favorites[veg]  
> "carrot"
```

property name

SETTING A PROPERTY VALUE WITH SQUARE BRACKET NOTATION

object

```
let favorites = {  
  fruit: "apple",  
  veg: "carrot",  
  declare: function() {  
    console.log("I like fruit and veg");  
  }  
}
```

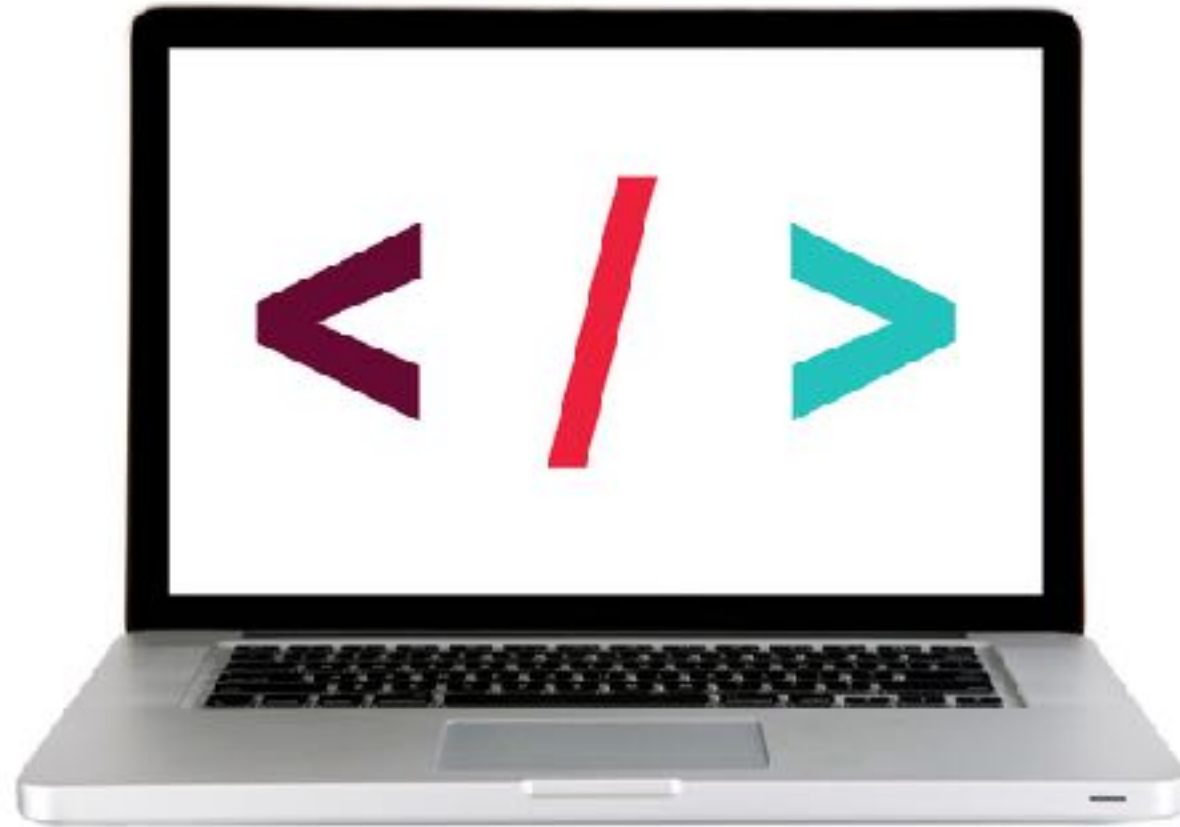
setting properties

```
favorites[fungus] = 'shiitake';  
favorites[pet] = 'hamster';
```

setting a method

```
favorites[beAmbivalent] = function() {  
  console.log("I like other things");  
};
```

LET'S TAKE A CLOSER LOOK



EXERCISE — OBJECTS



EXERCISE

KEY OBJECTIVE

- ▶ Create JavaScript objects using object literal notation

TYPE OF EXERCISE

- ▶ Groups of 2-3 (same group as for previous exercise)

TIMING

3 min

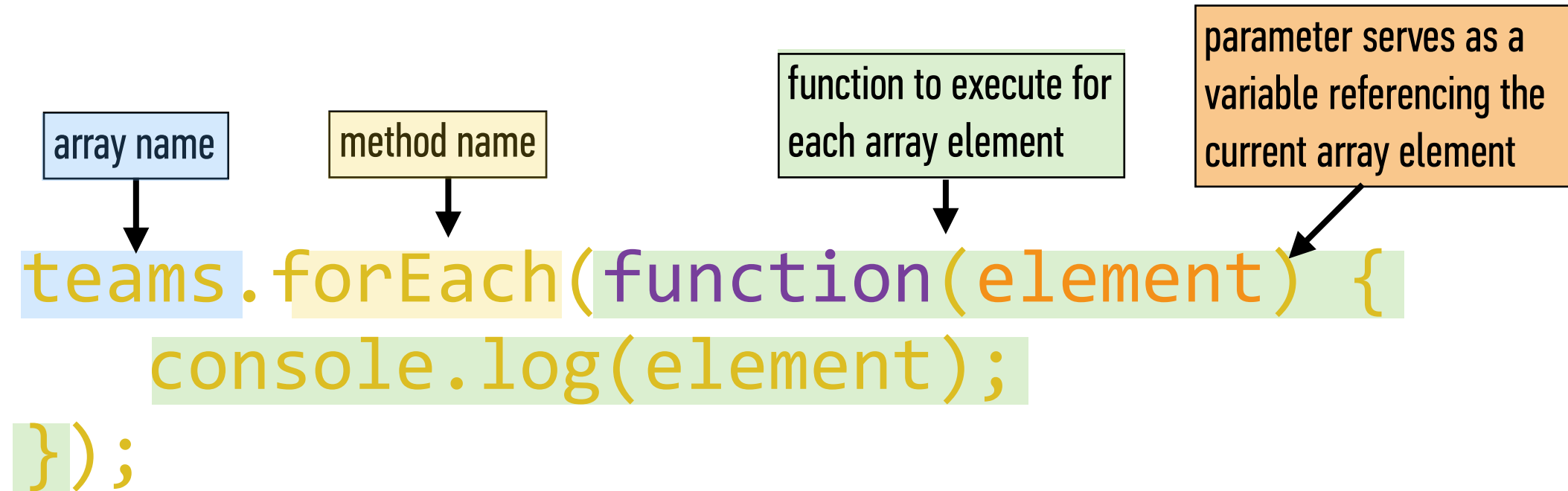
1. On your desk or on the wall, write code to create a variable whose name corresponds to the thing you were assigned in the previous exercise (cloud, houseplant, nation, office chair, or airplane).
2. Write code to add a property to the object and specify a value for the property.
3. Write code to add a method to the object, and specify a value for the method (use a comment or `console.log()` statement for the function body).
4. BONUS: Rewrite your answers for 1-3 as a single JavaScript statement.

ARRAY ITERATOR METHODS

ARRAY ITERATOR METHODS

forEach()	Executes a provided function once per array element
every()	Tests whether all elements in the array pass the test implemented by the provided function
some()	Tests whether some element in the array passes the text implemented by the provided function
filter()	Creates a new array with all elements that pass the test implemented by the provided function
map()	Creates a new array with the results of calling a provided function on every element in this array

forEach()



forEach() EXAMPLE

```
let teams = ['Bruins', 'Bears', 'Ravens', 'Ducks'];
```

```
teams.forEach(function(element) {  
    console.log(element);  
});
```


REAL WORLD SCENARIOS

REAL WORLD SCENARIO

A user, browsing on a shopping website, searches for size 12 running shoes, and examines several pairs before purchasing one.

OBJECTS = NOUNS

A **user**, browsing on a **shopping website**, searches for size 12 running shoes, and examines **several pairs** before purchasing one.

implicit object:

shopping cart

PROPERTIES = ADJECTIVES

A user, browsing on a shopping website, searches for **size 12** **running** shoes, and examines several pairs before purchasing one.

implicit properties:

for each pair of shoes:

price
color

for the shopping cart:

contents
total
shipping
tax

METHODS = VERBS

A user, browsing on a shopping website, **searches** for size 12 running shoes, and examines several pairs before purchasing one.

implicit methods:

for each pair of shoes:

add to cart

for the shopping cart:

**calculate shipping
calculate tax
complete purchase
remove item**

EXERCISE — REAL WORLD SCENARIOS & OBJECTS



EXERCISE

KEY OBJECTIVE

- ▶ Identify likely objects, properties, and methods in real-world scenarios

TYPE OF EXERCISE

- ▶ Groups of 3-4

TIMING

10 min

1. Read through your scenario together.
2. Identify and write down likely objects, properties, and methods in your scenario. (Remember to consider implicit objects as well as explicit ones.)
3. Choose someone to report your results to the class.

LAB — OBJECTS



KEY OBJECTIVE

- ▶ Create JavaScript objects using object literal notation

TYPE OF EXERCISE

- ▶ Individual or pair

TIMING

20 min

1. Open starter-code > 4-object-exercise > monkey.js in your editor.
2. Create objects for 3 different monkeys each with the properties and methods listed in the start file.
3. Practice retrieving properties and using methods with both dot notation and bracket syntax.
4. BONUS: Rewrite your code to use a constructor function.

Exit Tickets!

(Class #4)

LEARNING OBJECTIVES – REVIEW

- Determine the scope of local and global variables
- Describe what hoisting does
- Identify likely objects, properties, and methods in real-world scenarios
- Create JavaScript objects using object literal notation

NEXT CLASS PREVIEW

Slack Bot Lab

- › Install and configure all utilities needed to build a bot using the Hubot framework
- › Write scripts that allow your bot to interact with users of the class Slack organization

Q&A