GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `vodnik/JS-SF-16-resources` repo to your computer:
   - Open the terminal
   - `cd` to the `~/Documents/JSD/JS-SF-16-resources` directory
   - Type **`git pull`** and press **return**
2. In your editor, open the following folder:
   `Documents/JSD/JS-SF-16-resources/04-scope-objects`

SCOPE & OBJECTS

LEARNING OBJECTIVES

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

- Create and call a function that accepts parameters to solve a problem
- Return a value from a function using the `return` keyword
- 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

- Function parameters
- The `return` keyword
- Variable scope
- The `var`, `let`, and `const` keywords
- Hoisting
- Objects

WEEKLY OVERVIEW

WEEK 3	Scope & objects / Slack Bot Lab
WEEK 4	JSON & Intro to DOM / DOM & jQuery
WEEK 5	Advanced jQuery / Ajax & APIs

EXIT TICKET QUESTIONS

HOMEWORK REVIEW

HOMEWORK — GROUP DISCUSSION

EXERCISE

TYPE OF EXERCISE

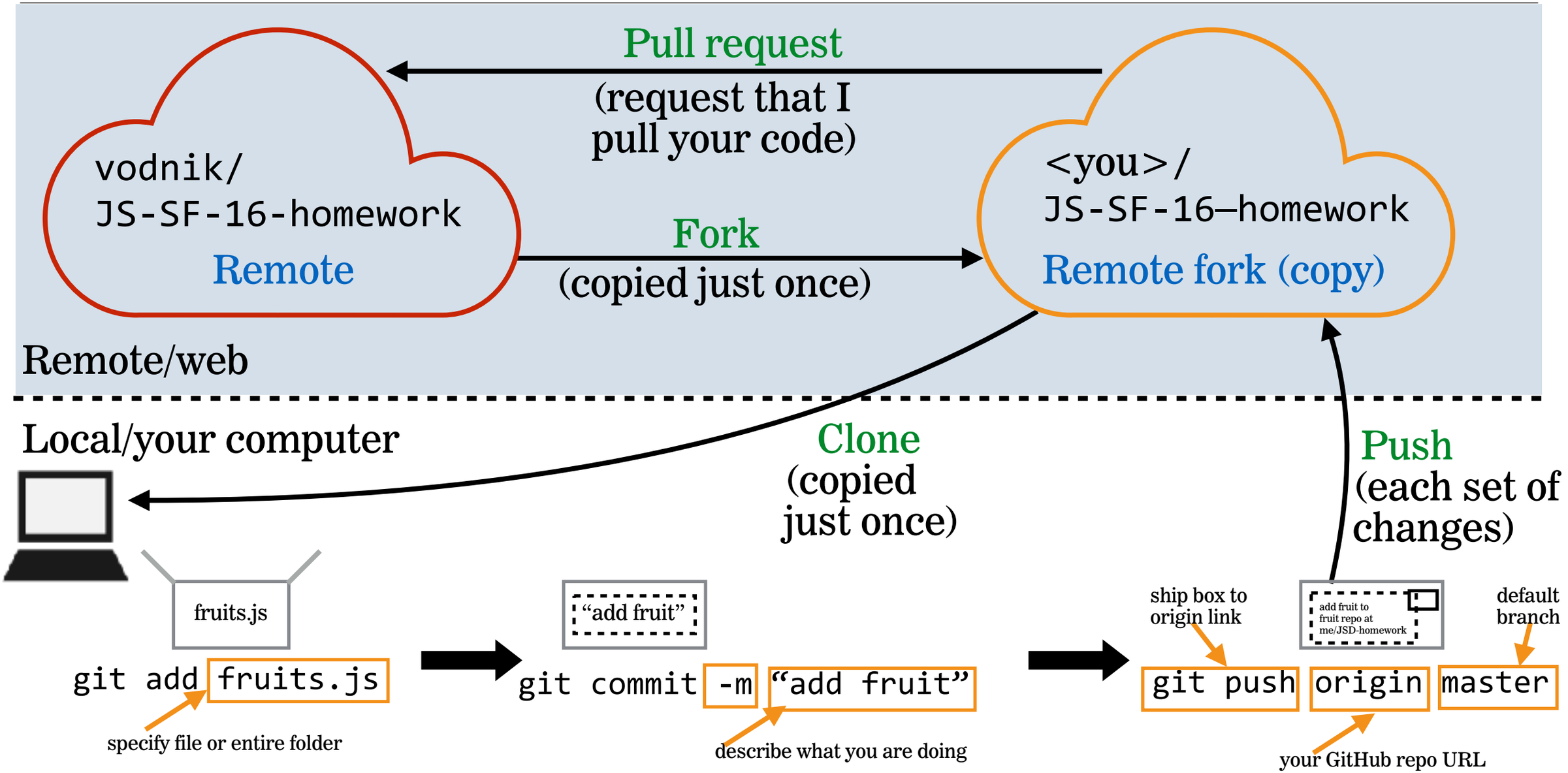
- Groups of 2-3

TIMING

5 min

1. Take turns showing and explaining your code.
2. Share 1 thing you're excited about being able to accomplish.
3. Have each person in the group note 1 thing they found challenging for the homework. Discuss as a group how you think you could solve each problem.
4. Did you work on the Madlibs bonus exercise? Show your group what you did!

USING THE JS-SF-16-HOMEWORK REPO

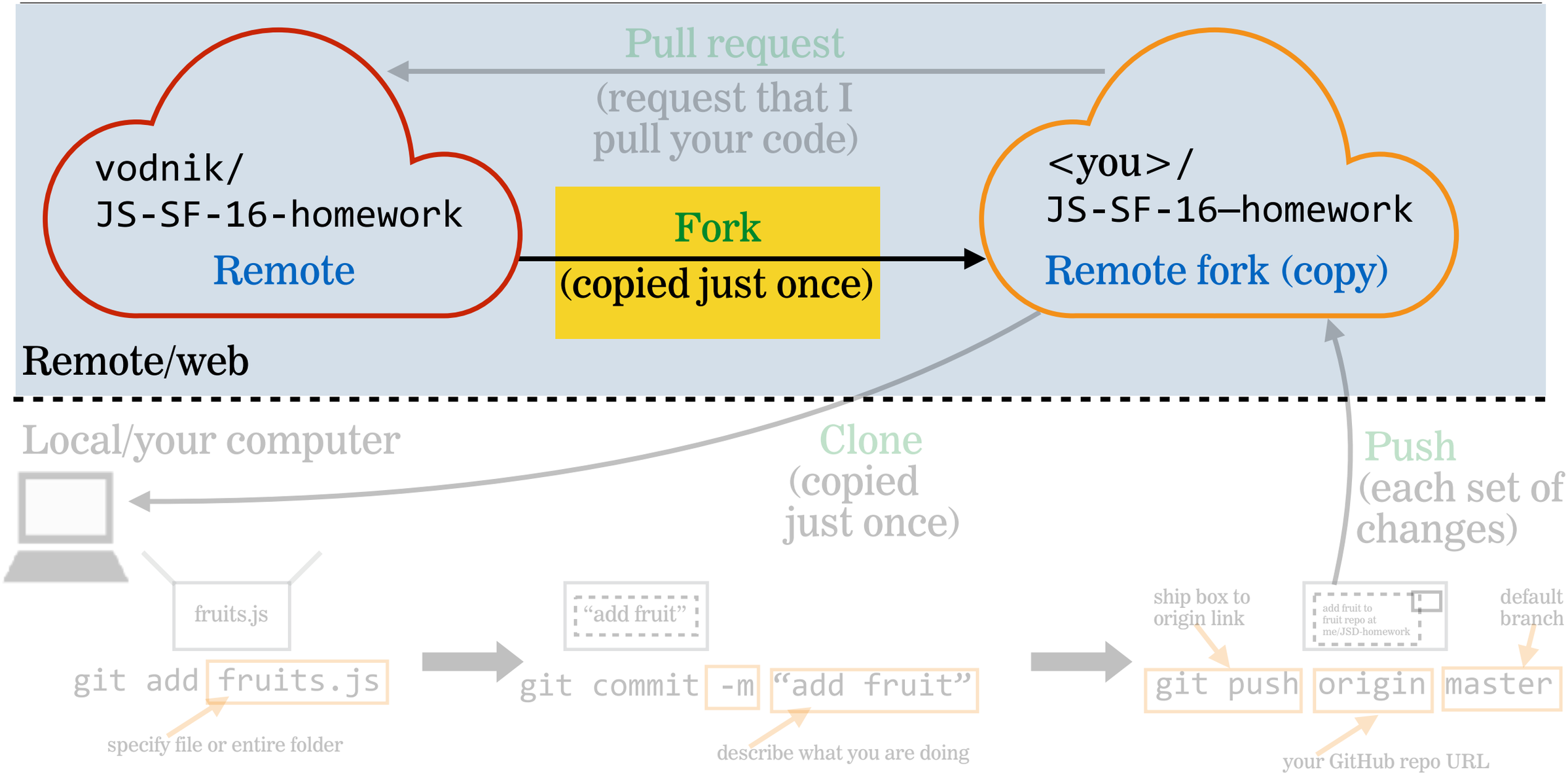


SUBMIT HOMEWORK: SETUP (ONE TIME ONLY)

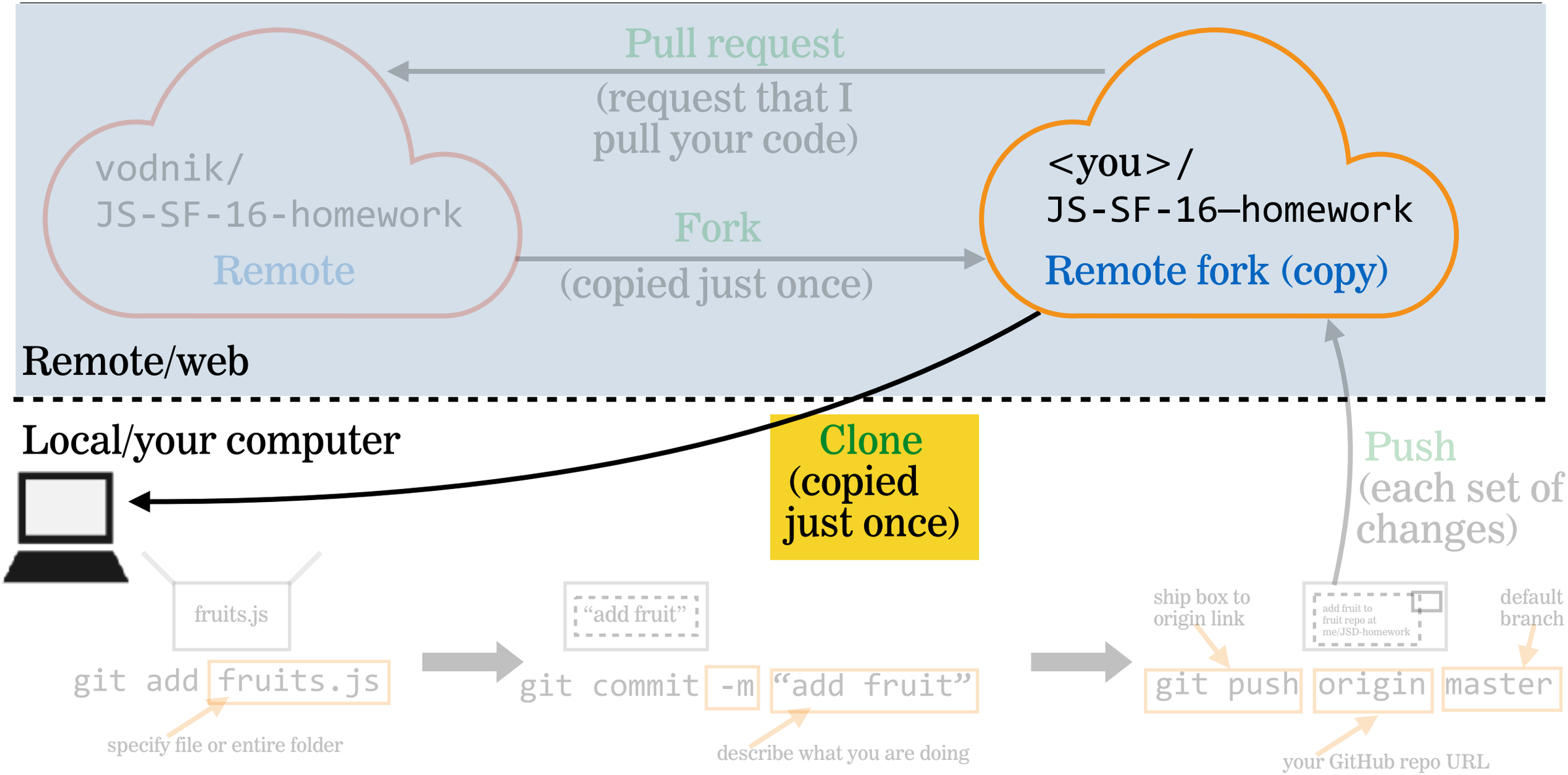
On github.com:

- Open https://git.generalassemb.ly/vodnik/JS-SF-16-homework
- Fork this repo to your GitHub account
- Clone your fork to your computer, within your JSD folder

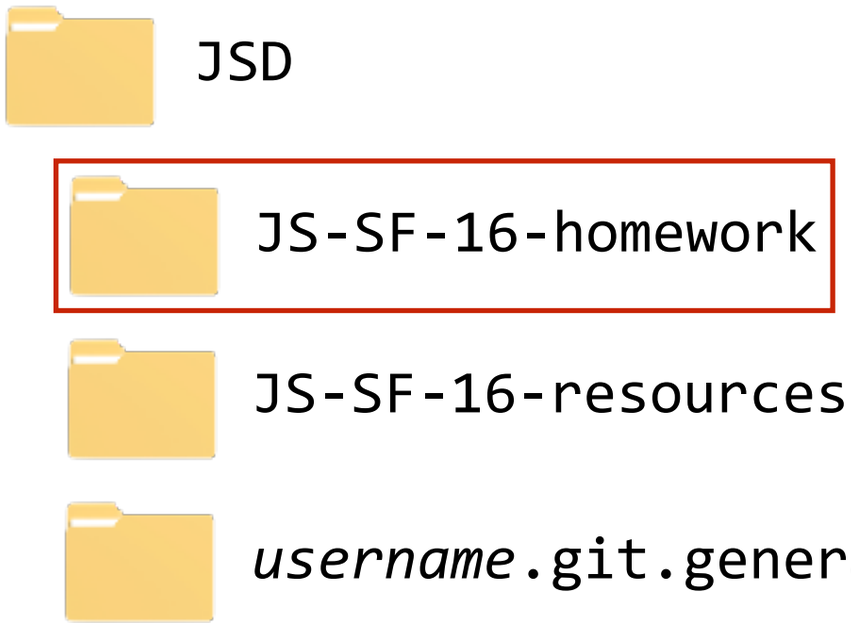
USING THE JS-SF-16-HOMEWORK REPO



USING THE JS-SF-16-HOMEWORK REPO



HOMEWORK FOLDER LOCATION

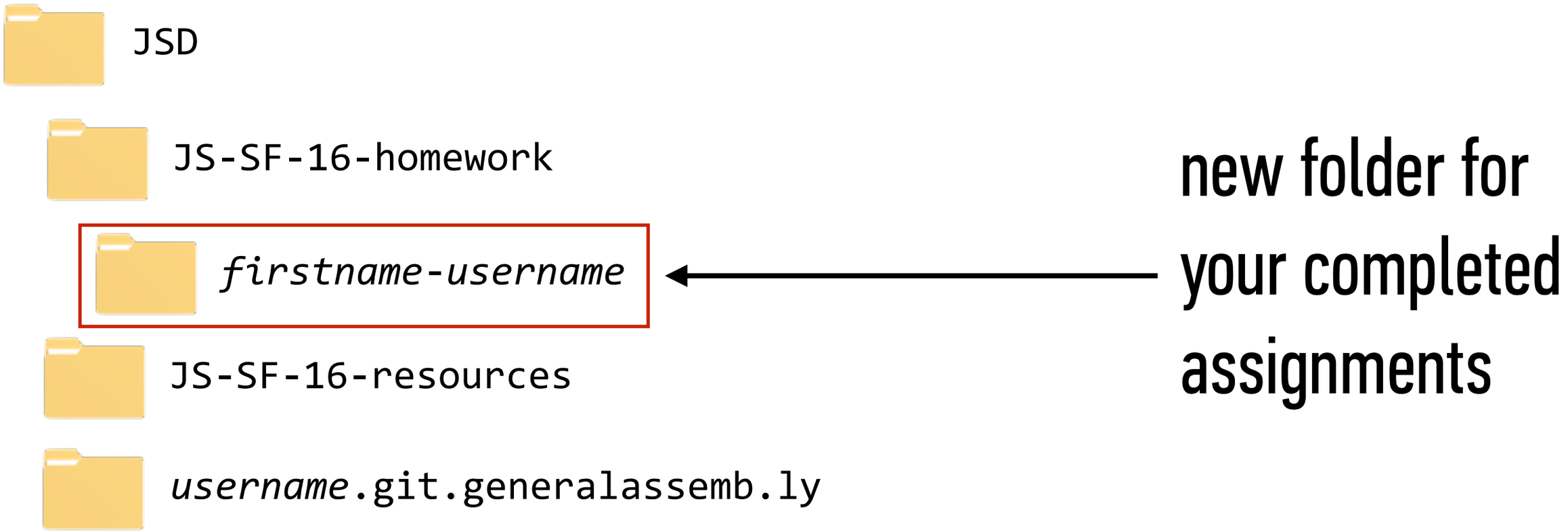


new folder for your clone of the homework repo

SUBMIT HOMEWORK: SETUP (CONTINUED)

- Within your new `JS-SF-16-homework` folder, create a new subfolder and name it your first name, a hyphen, and your github name. For instance, Sasha's folder would be `Sasha-vodnik`.

PERSONAL ASSIGNMENTS FOLDER LOCATION



SETUP DONE!

- Reminder: Now that you've completed the preceding setup, you never have to do it again!
- Each time you submit homework for the rest of this course, you'll repeat only the steps that follow.

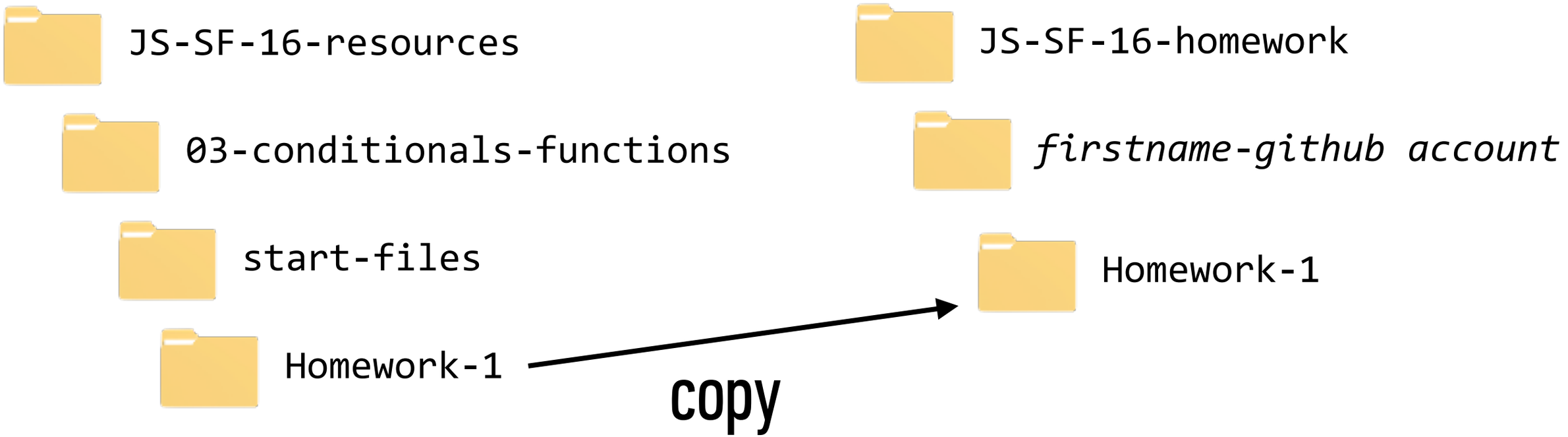
SUBMIT HOMEWORK: STEP 1

THIS IS THE START OF STEPS YOU'LL DO TO SUBMIT EVERY HOMEWORK ASSIGNMENT

In Finder:

- navigate to *`firstname-username`* folder (example: `Sasha-vodnik`)
- copy your completed `Homework-1` folder from last Wednesday into your *`firstname-username`* folder.

SUBMIT HOMEWORK: STEP 1 ILLUSTRATION

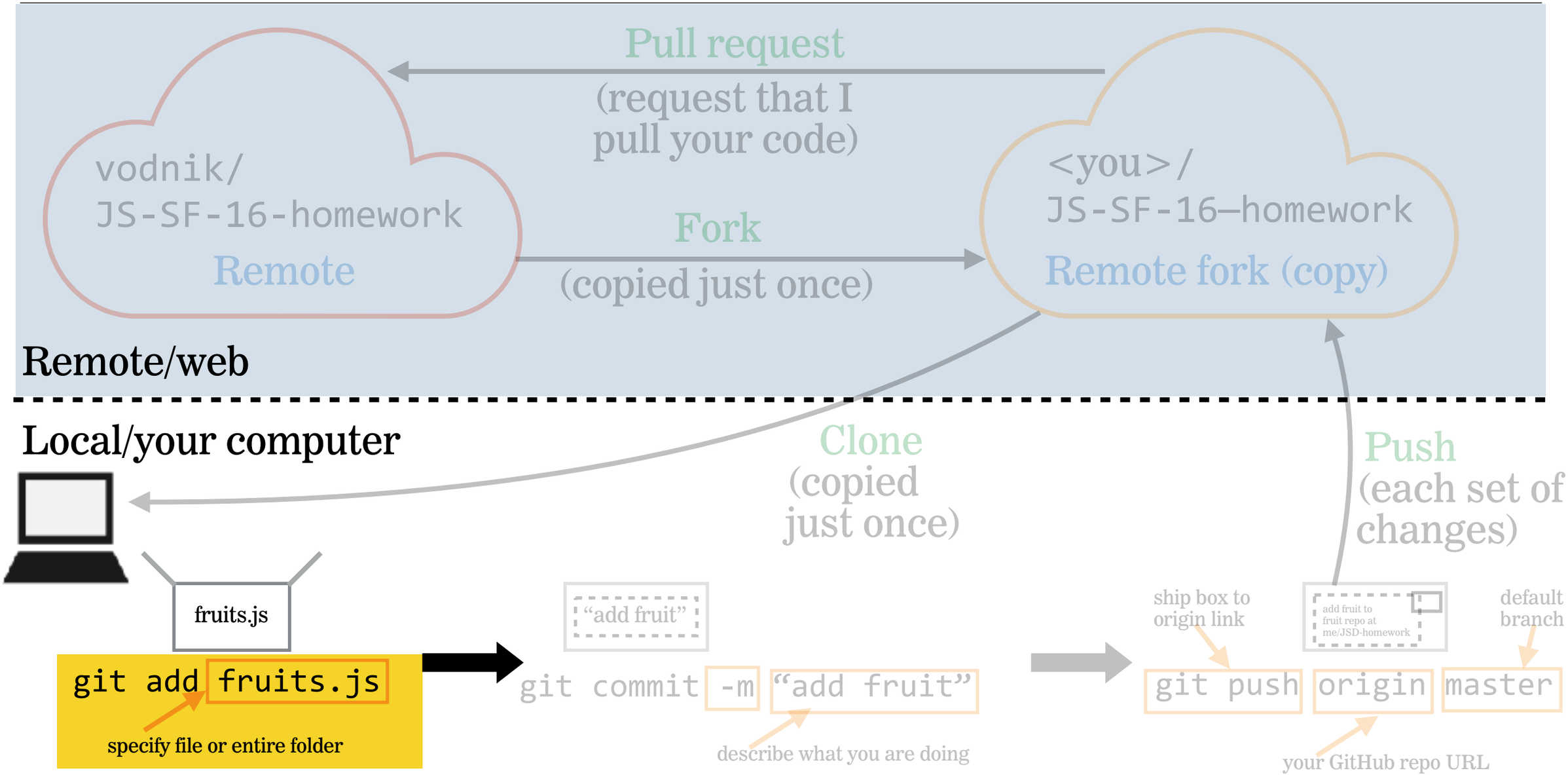


SUBMIT HOMEWORK: STEP 2

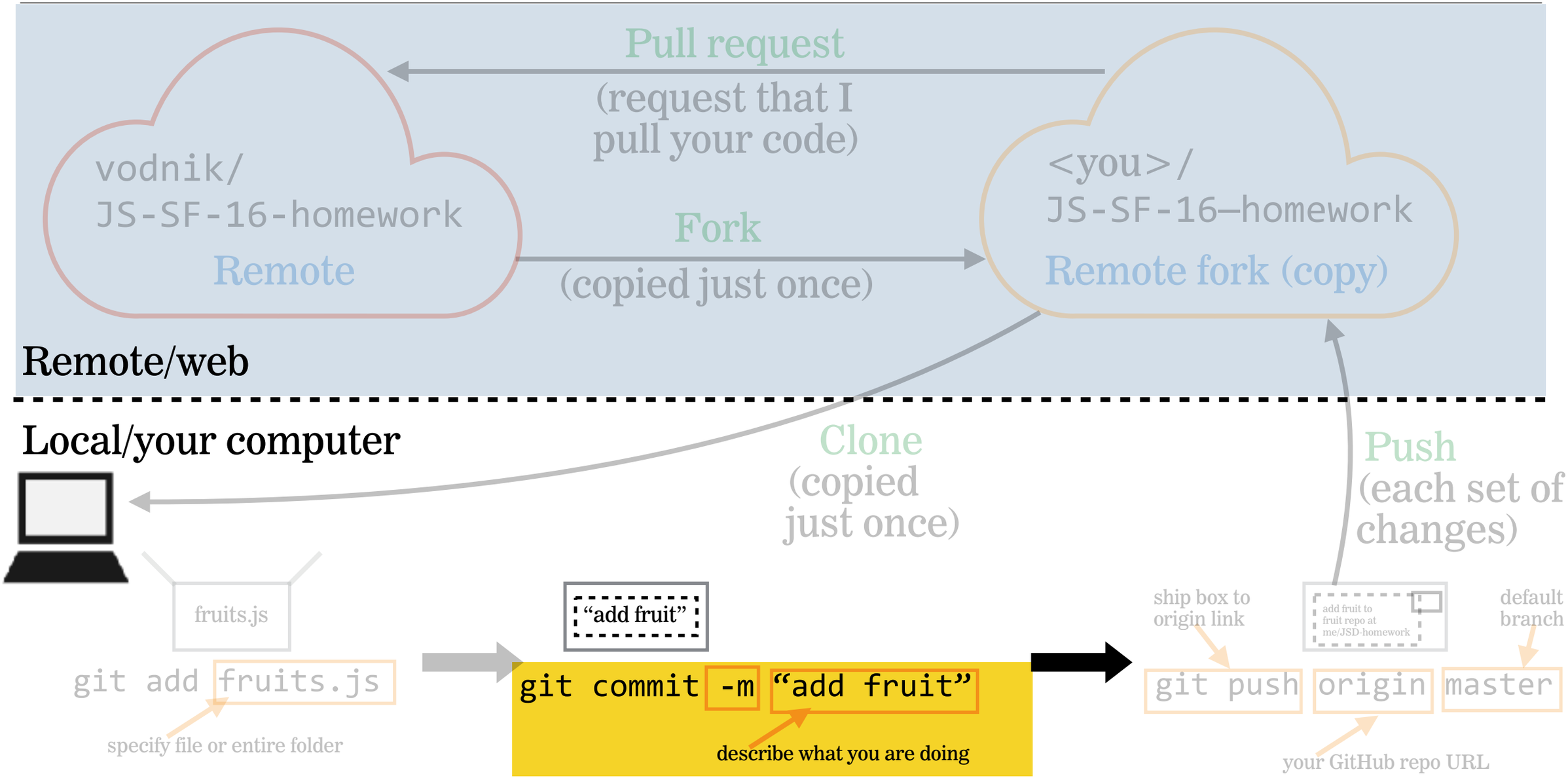
In Terminal:

- navigate to `JS-SF-16-homework` folder
- `git add .`
- `git commit -m “submitting Homework 1”`
- `git push origin master`

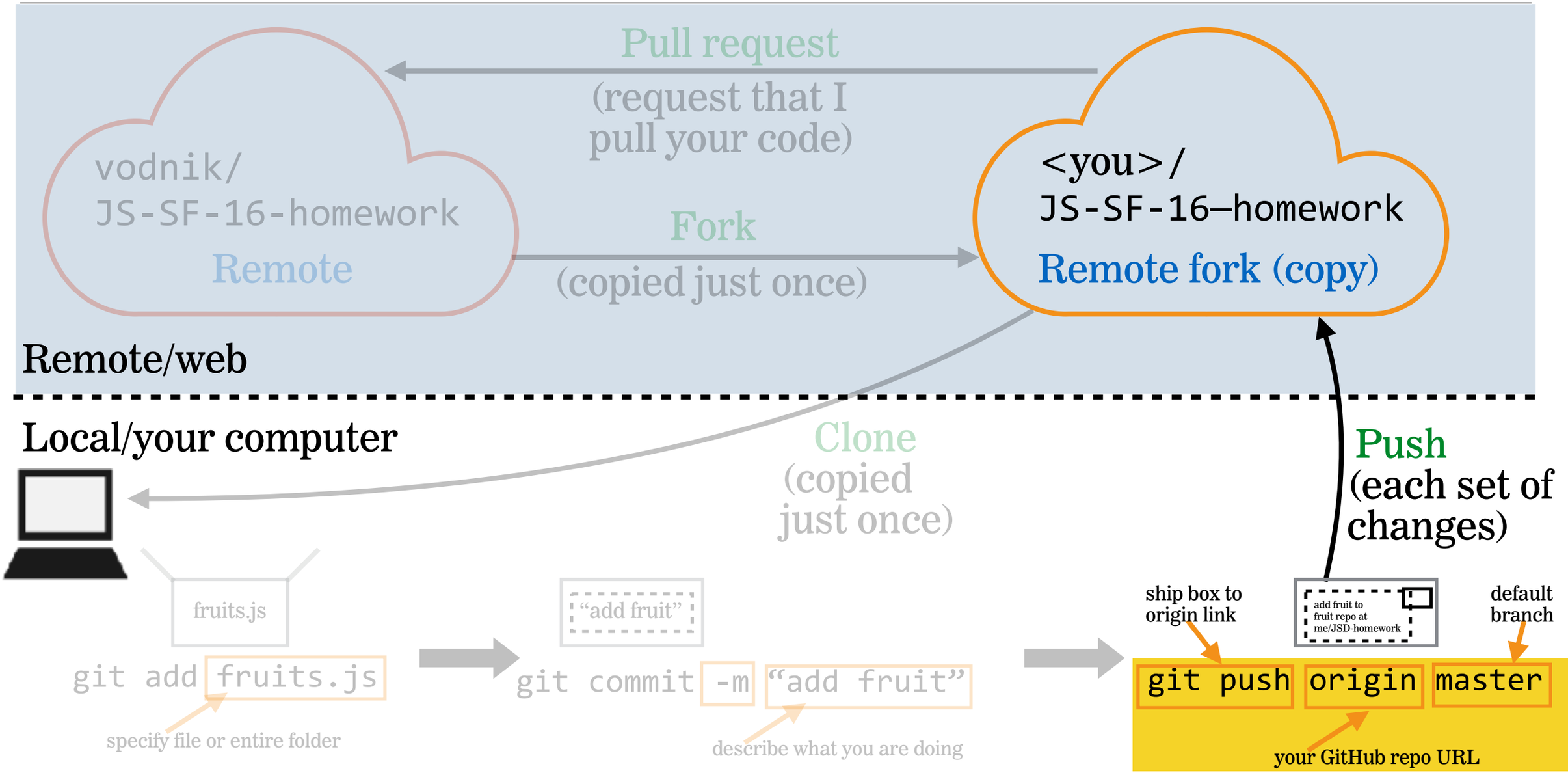
USING THE JS-SF-16-HOMEWORK REPO



USING THE JS-SF-16-HOMEWORK REPO



USING THE JS-SF-16-HOMEWORK REPO

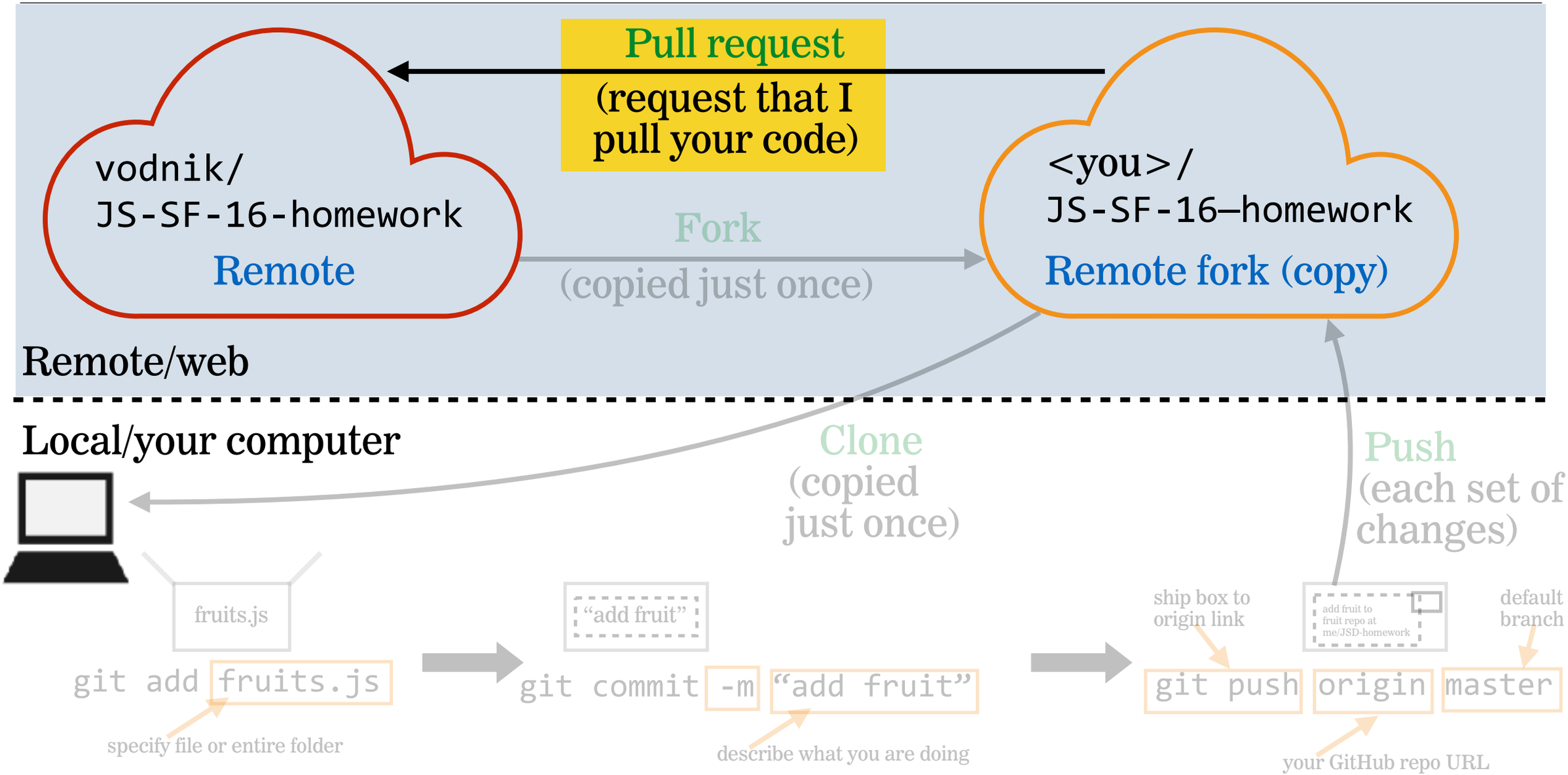


SUBMIT HOMEWORK: STEP 3

In Browser:

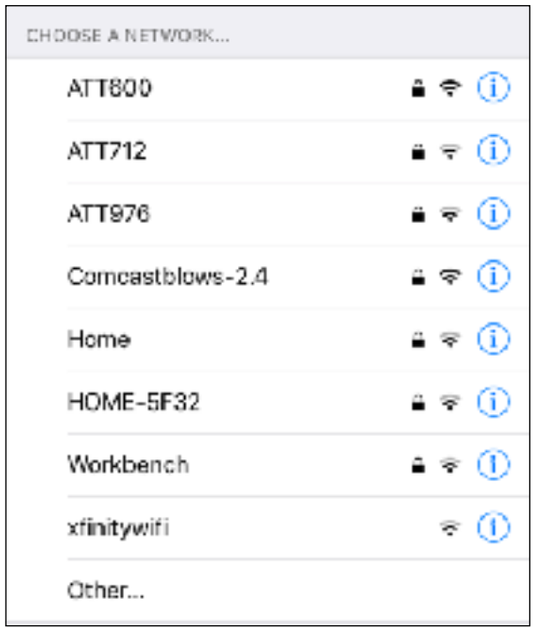
- Go to your fork of `JS-SF-16-homework` on `git.generalassemb.ly`
- click **New pull request**
- click **Create pull request**
- click **Create pull request** (again)

USING THE JS-SF-16-HOMEWORK REPO



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

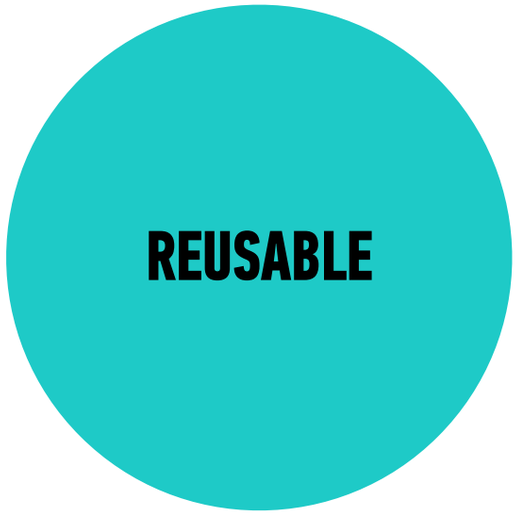
- home
- GA
- in a car
- on BART/MUNI



FUNCTIONS

FUNCTIONS

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



We can use the same function multiple times

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

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!”);
};
```

CALLING A FUNCTION

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

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

```
pickADescriptiveName();
```

Function name + parentheses

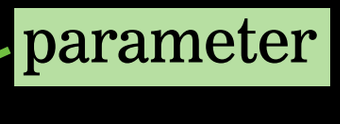
PARAMETERS

DOES THIS CODE SCALE?

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

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

USING A PARAMETER




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


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

sayHello('Otto');
=> 'Hello Otto’
```

USING MULTIPLE PARAMETERS


```
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

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 > 6-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

KEY OBJECTIVE

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

TYPE OF EXERCISE

- Groups of 2 - 3

LOCATION

- `starter-code > 6-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!

THE `return` STATEMENT

`return` STATEMENT

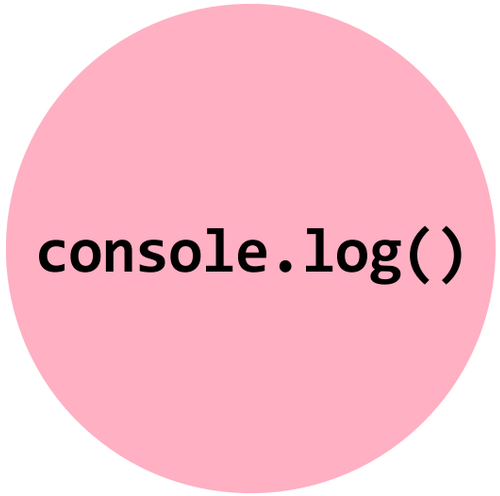
- Ends function's execution
- Returns a value — the result of running the function

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`



vs

- 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

KEY OBJECTIVE

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

TYPE OF EXERCISE

- Individual or pair

LOCATION

- `starter-code > 7-price-calculator`

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 1: Convert your function to assume a currency of "dollar" by default.
3. BONUS 2: Convert your code to use arrow functions.

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

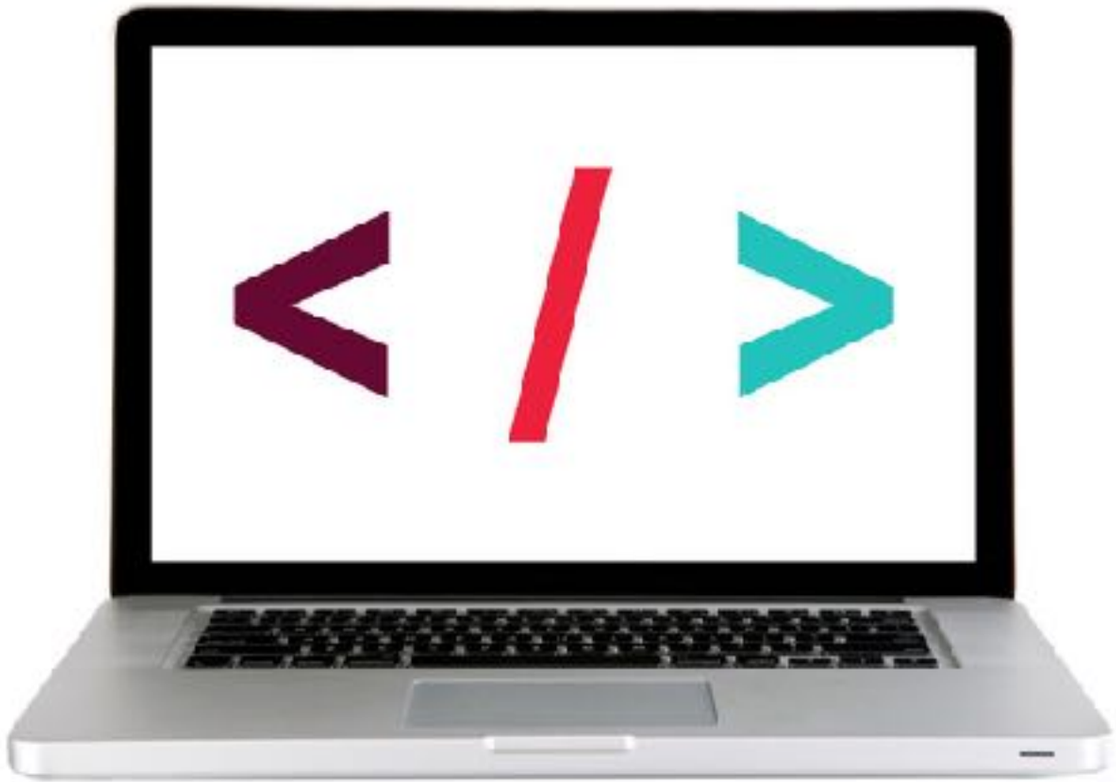
- A variable created with `let` or `const` creates local scope within any block, including blocks that are part of loops and conditionals.
- This is known as **block scope**, which is another type of local scope.

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

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

```
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
```

LET'S TAKE A CLOSER LOOK



EXERCISE — SCOPE

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)

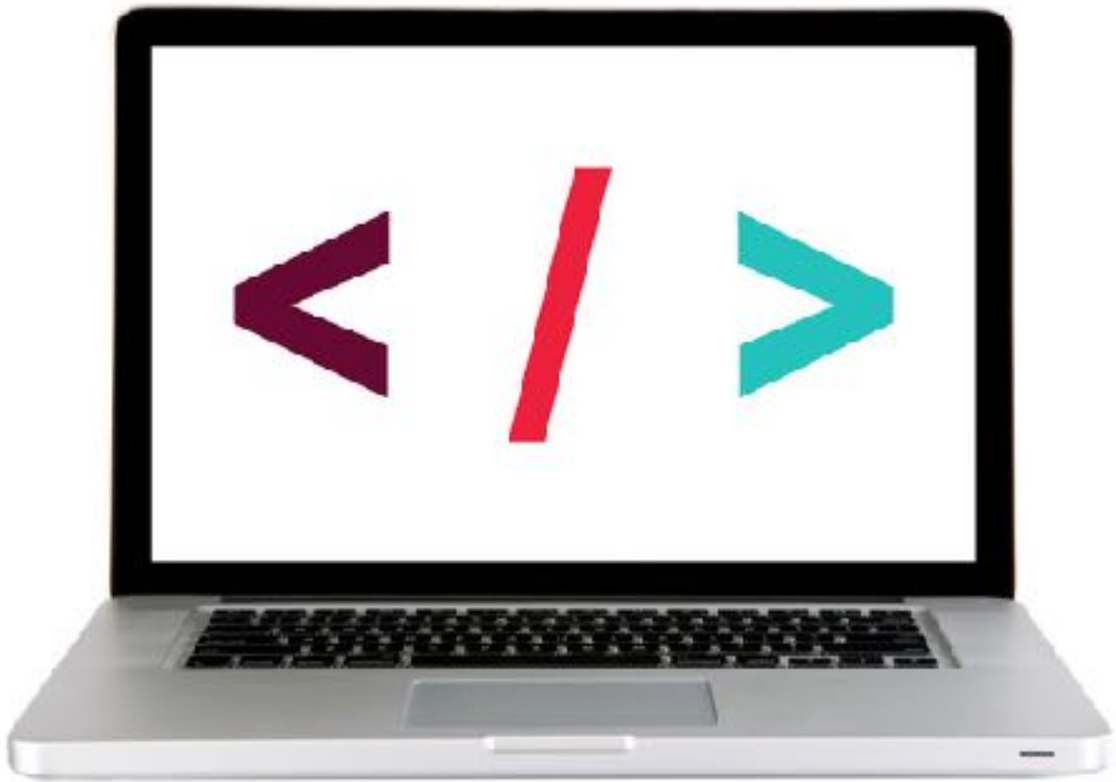
```
let results = [0,5,2];
```

const

- newer keyword (ES6)
- local scope within functions **and** within any block (including loops and conditionals)
- used to declare constants
  - » once you've declared a value using `const`, you can't reassign the value in that scope
  - » by contrast, variables declared with `var` or `let` can be reassigned to other values

```
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

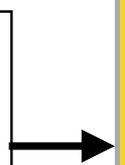
```
var results = [0,5,2];
```

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
```

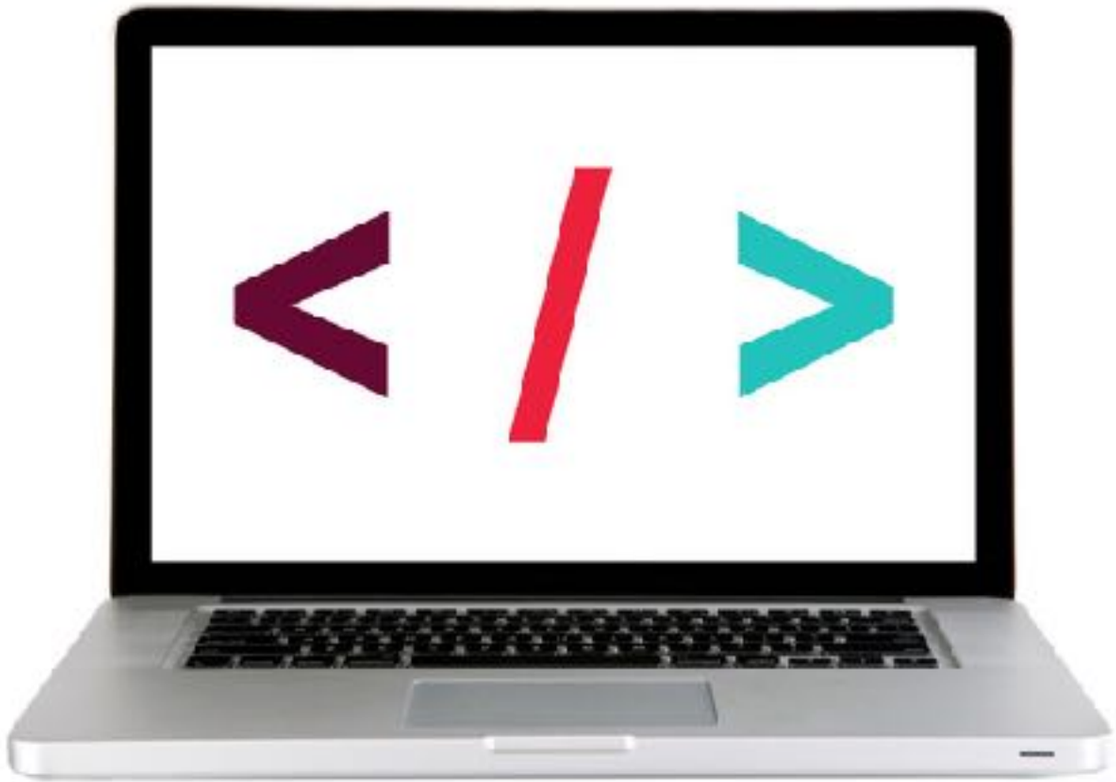
`let`, `const`, `var`, AND BROWSER SUPPORT

- `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 reassign 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

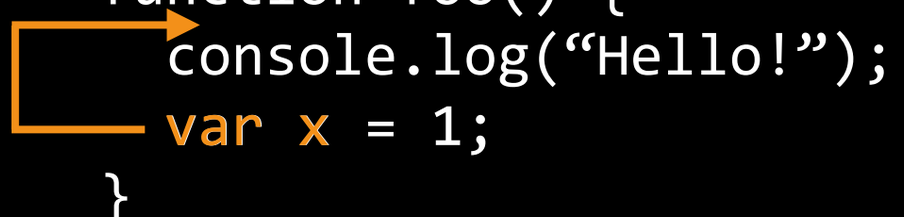
1. Open the index.html file in your browser, view the console, and examine the error.
2. Follow the instructions in `js > app.js` to complete parts A and B.

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!”);
}
```

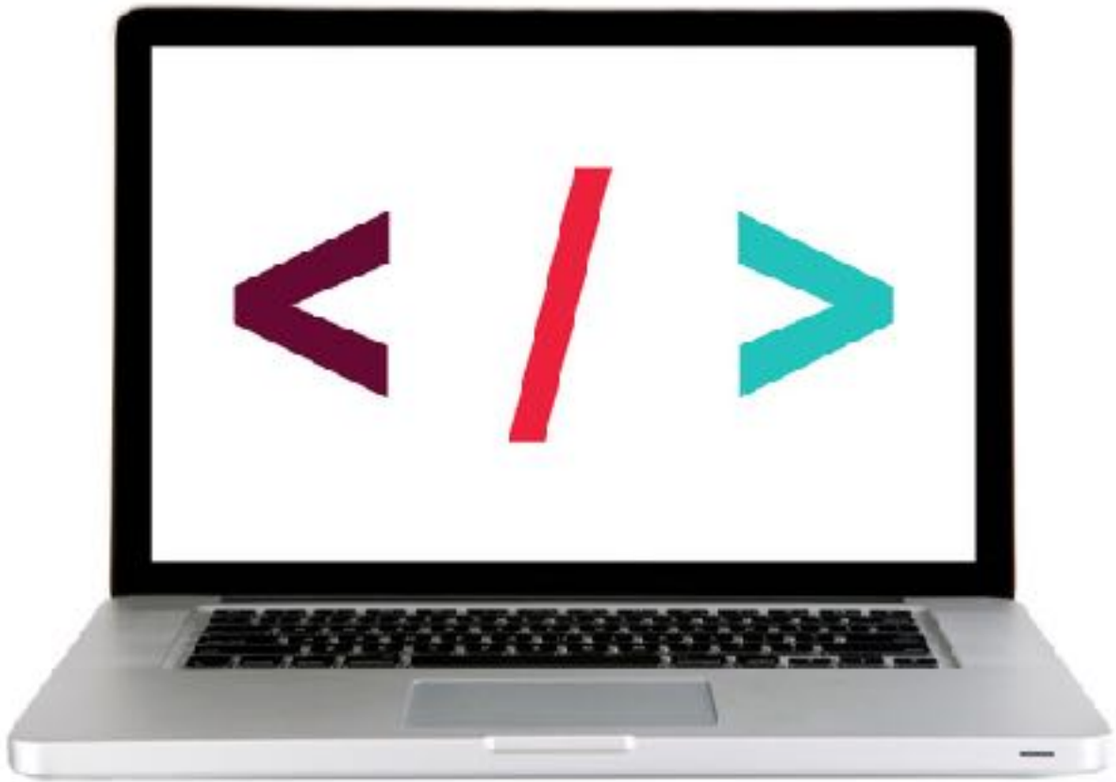
VARIABLES AND HOISTING

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

FUNCTIONS AND HOISTING

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

LET'S TAKE A CLOSER LOOK



EXERCISE — HOISTING

KEY OBJECTIVE

- Describe what hoisting does

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

KEY OBJECTIVE

- Create JavaScript objects using object literal notation

TYPE OF EXERCISE

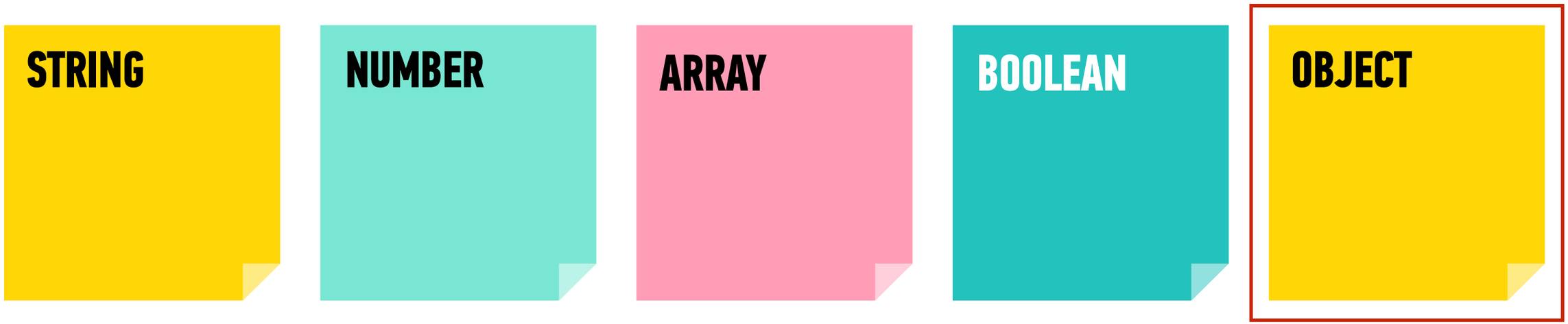
- Groups of 2-3

TIMING

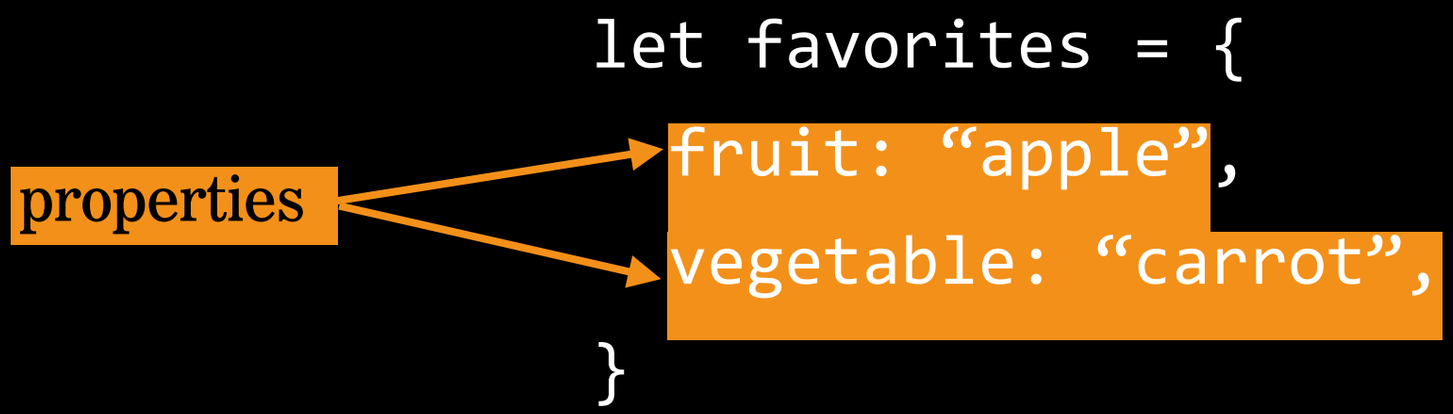
3 min

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



AN OBJECT IS A COLLECTION OF PROPERTIES



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”
}
```

AN OBJECT IS NOT ORDERED

```
0  [
     “apple”,
1    “pear”,
2    “banana”
   ]
```

ARRAY
ordered

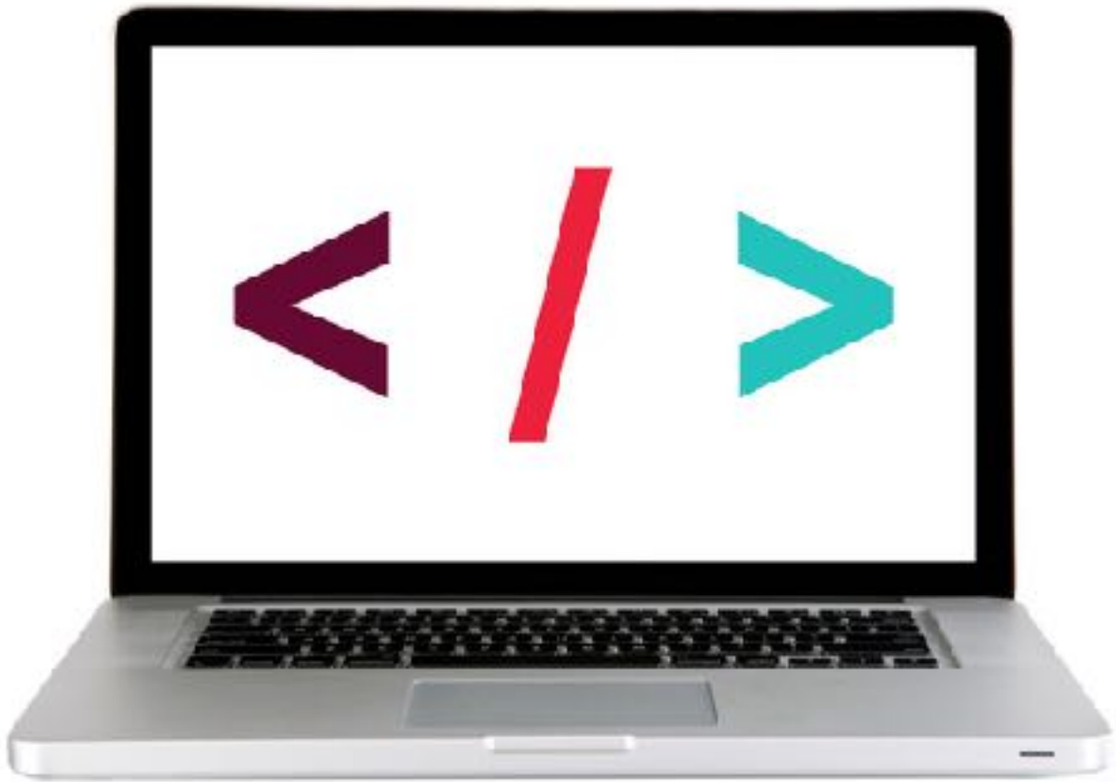
```
{
  fruit: “apple”,
  vegetable: “carrot”,
  fungus: “trumpet mushroom”
}
```

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!”);
  },
}
```

LET'S TAKE A CLOSER LOOK



TWO WAYS TO GET/SET PROPERTIES

GETTING A PROPERTY VALUE WITH DOT NOTATION

object

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

getting properties

object name

```
favorites.fruit
> “apple”
favorites.veg
> “carrot”
```

property name

calling a method

object name

```
favorites.declare()
> “I like fruit and veg”
```

method name

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

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

getting properties

object name

property name

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

SETTING A PROPERTY VALUE WITH SQUARE BRACKET NOTATION

object

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

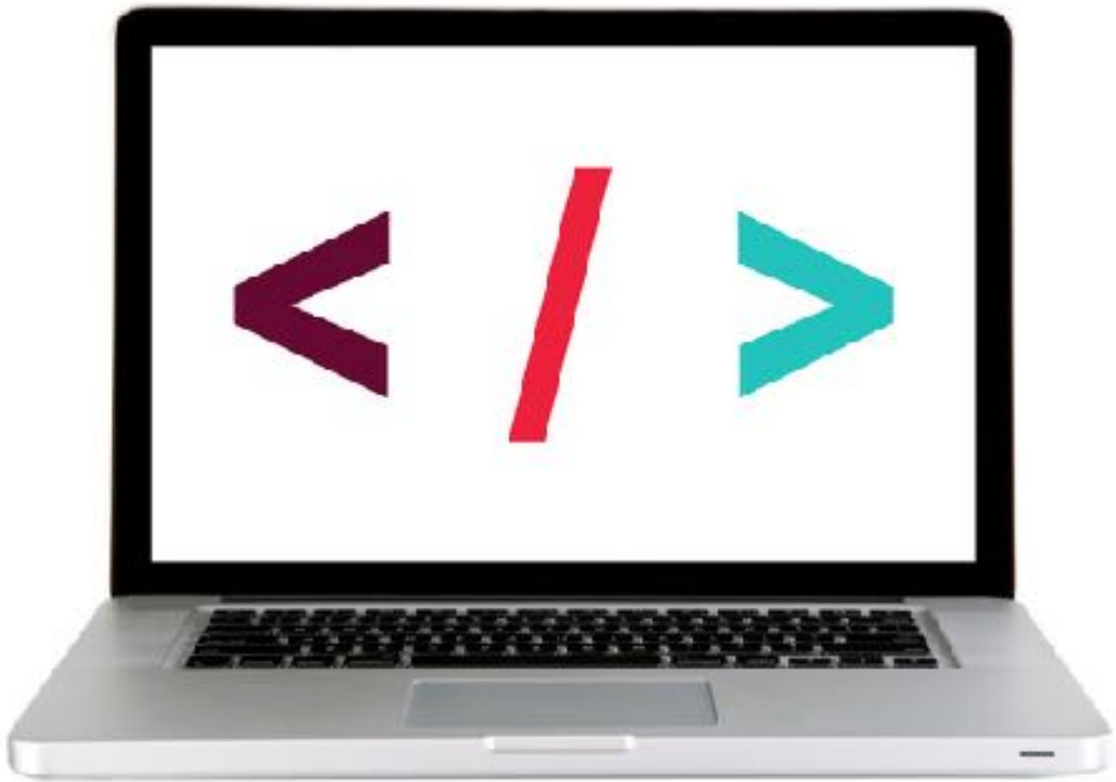
setting properties

```
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

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.

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

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

until 9:20

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

- Create and call a function that accepts parameters to solve a problem
- Return a value from a function using the `return` keyword
- 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