GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `vodnik/JS-SF-12-resources` repo to your computer:
   - Open the terminal
   - `cd` to the `~/Documents/JSD/JS-SF-12-resources` directory
   - Type **`git pull`** and press **return**
2. In your code editor, open the following folder: `Documents/JSD/JS-SF-12-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
- Create a program that hoists variables
- Identify likely objects, attributes, and methods in real-world scenarios
- Create JavaScript objects using object literal notation

AGENDA

- Set up homework repo & submit homework
- Variable scope
- The `var`, `let`, and `const` keywords
- Hoisting
- Objects

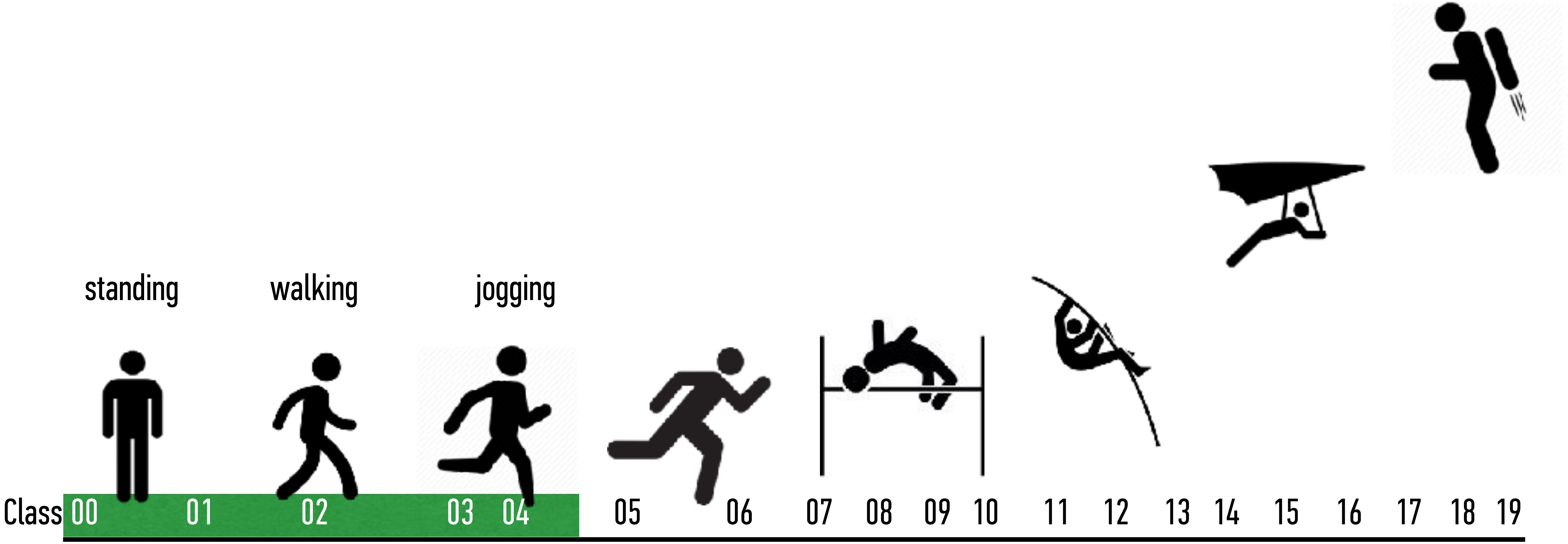
WEEKLY OVERVIEW

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

EXIT TICKET QUESTIONS

1. Hoisting. How does it work?
2. What is the difference between calling a function from inside another function and actually writing a function inside another function....? Like a nested function, and I know there is some funkyness with scope with these sorts of things functions calling functions
3. (suggestion:) More coding!
4. I prefer writing the code solo, then reviewing in a group

Where we are


standing
walking
jogging
Class 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19

SCOPE & OBJECTS

HOMEWORK REVIEW

HOMEWORK — GROUP DISCUSSION

EXERCISE

TYPE OF EXERCISE

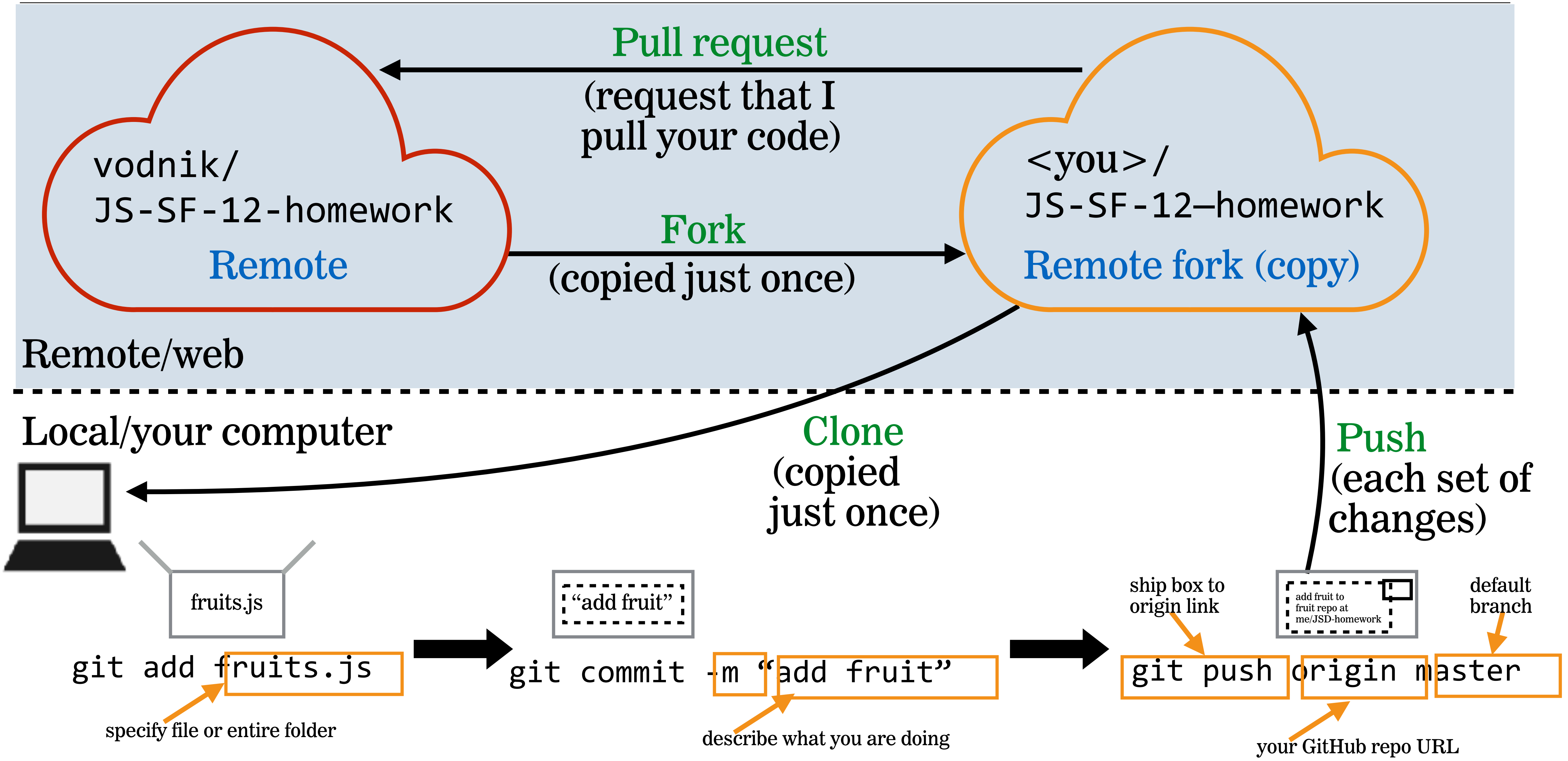
- Groups of 3

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 Random Address Generator bonus exercise? Show your group what you did!

USING THE JS-SF-12-HOMEWORK REPO



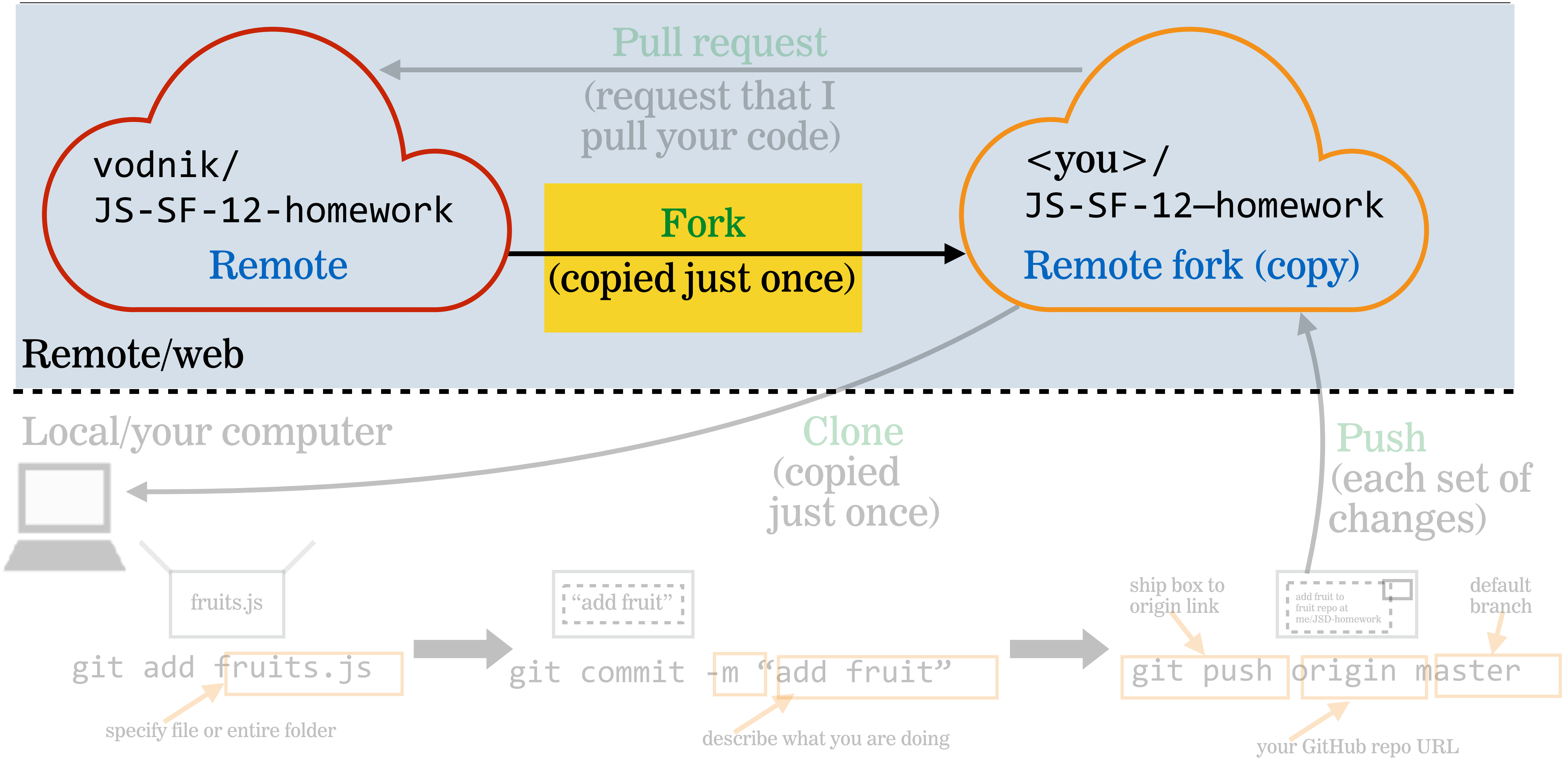
```
git add fruits.js
git commit -m "add fruit"
git push origin master
```

SUBMIT HOMEWORK: SETUP (ONE TIME ONLY)

On github.com:

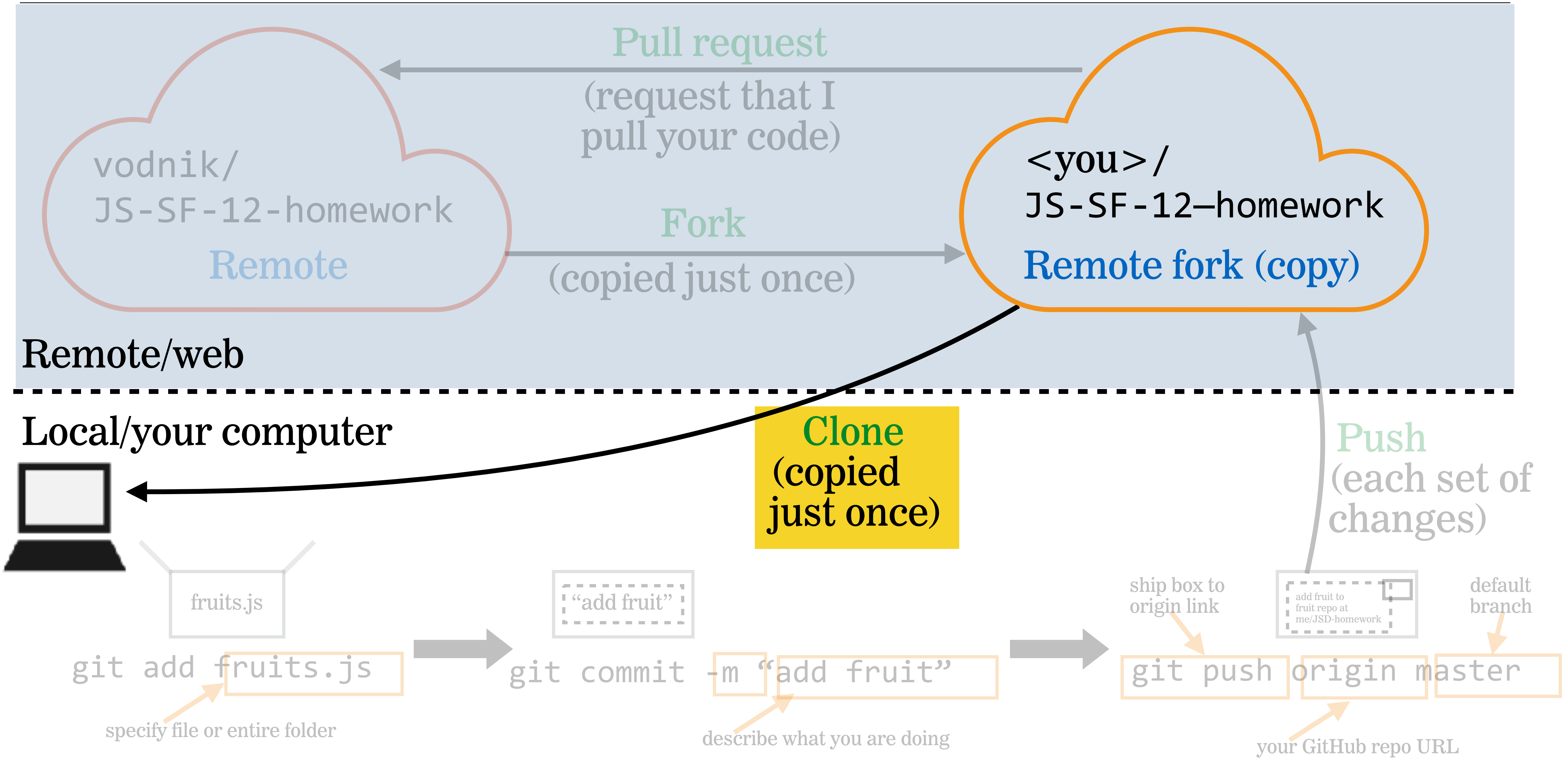
- Open https://git.generalassemb.ly/vodnik/JS-SF-12-homework
- Fork this repo to your GitHub account
- Clone your fork to your computer, within your JSD folder

USING THE JS-SF-12-HOMEWORK REPO

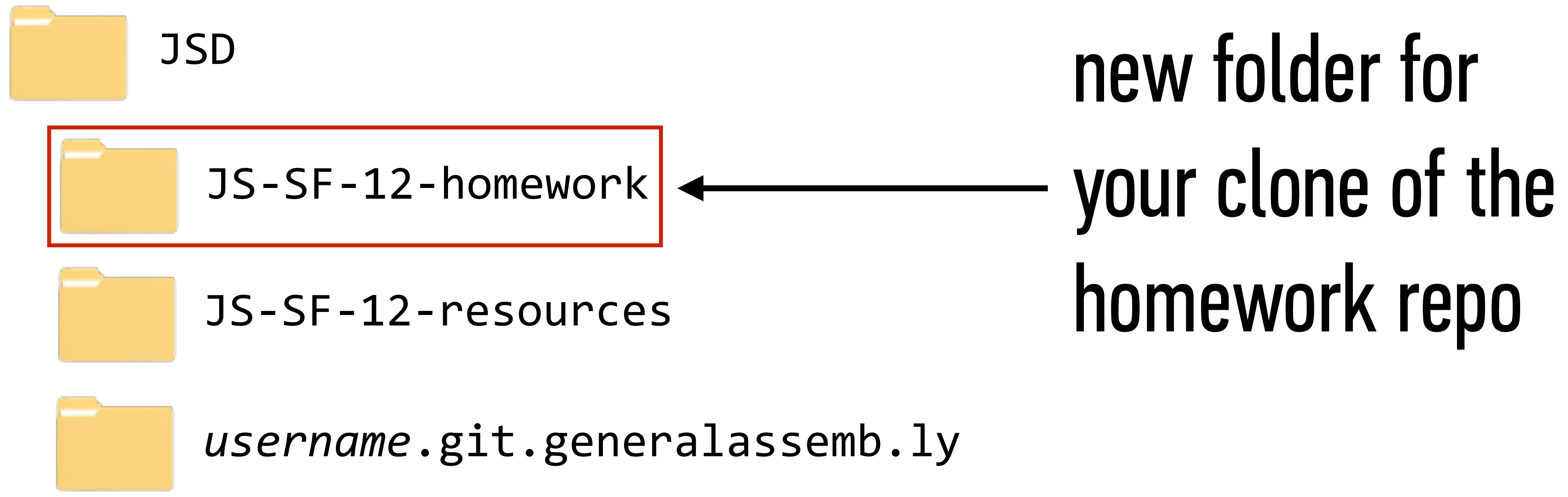


```
git add fruits.js
git commit -m "add fruit"
git push origin master
```

USING THE JS-SF-12-HOMEWORK REPO



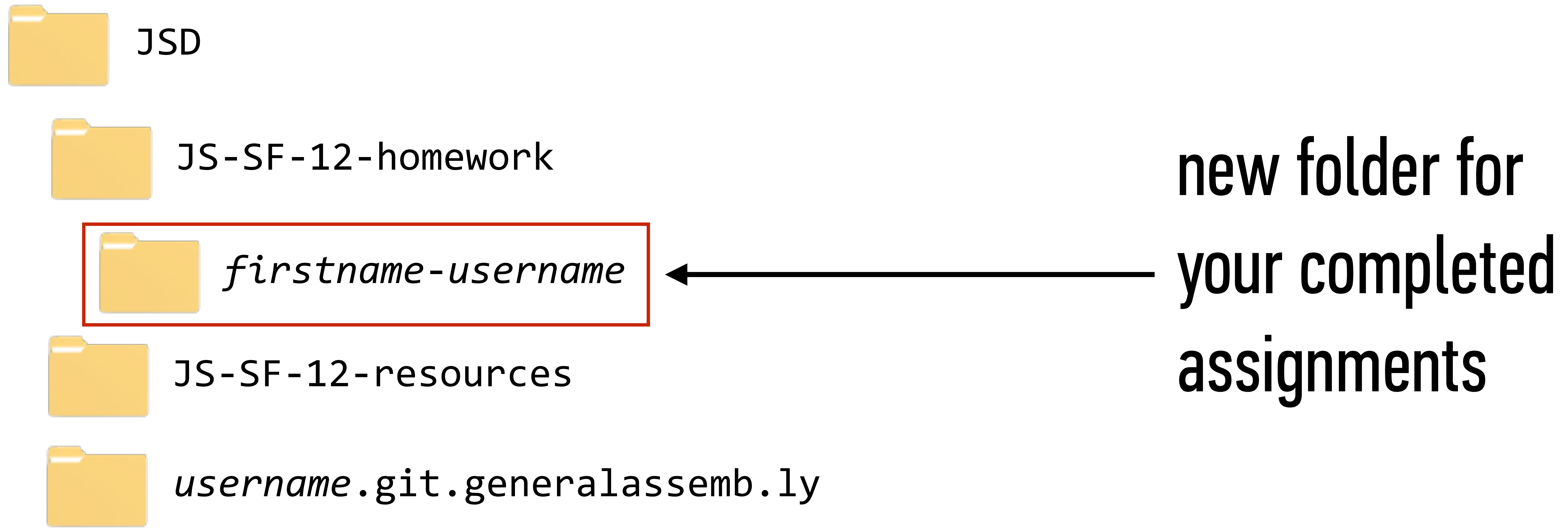
HOMEWORK FOLDER LOCATION


JSD
JS-SF-12-homework
JS-SF-12-resources
username.git.generalassemb.ly
new folder for your clone of the homework repo

SUBMIT HOMEWORK: SETUP (CONTINUED)

- Within your new **`JS-SF-12-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


JSD
JS-SF-12-homework
firstname-username
JS-SF-12-resources
username.git.generalassemb.ly
new folder for your completed assignments

SETUP DONE!

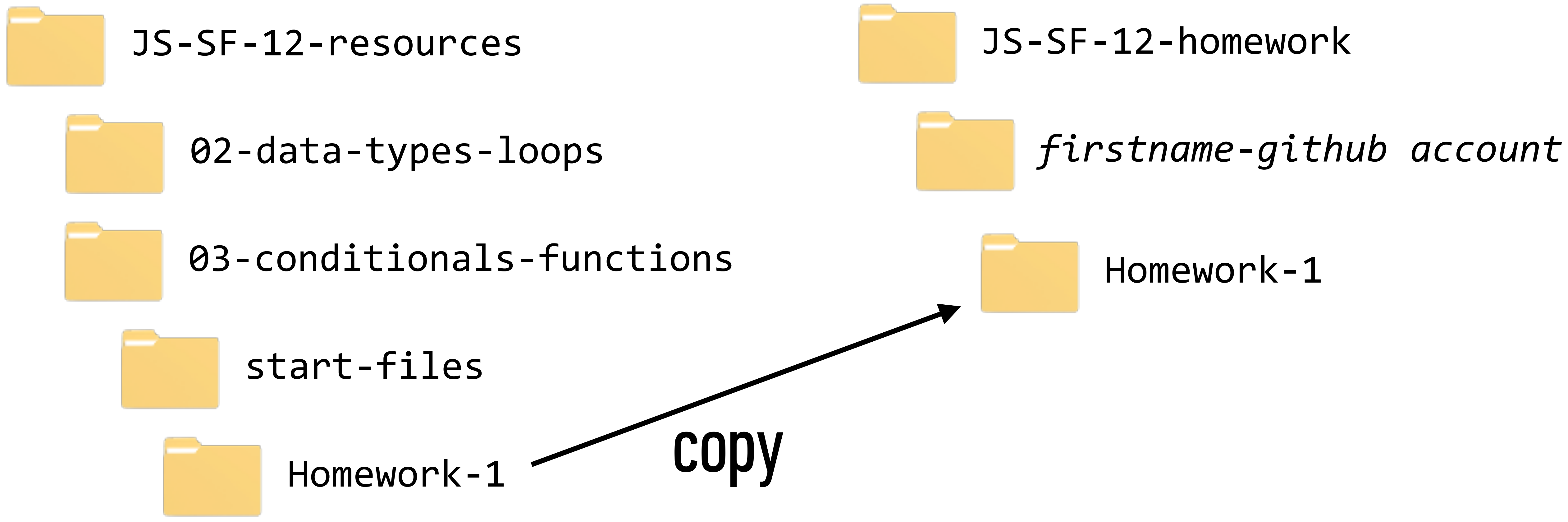
- Reminder: Now that you’ve completed the preceding setup, you never have to do it again!
- Each time you submit homework for the rest of this course, you’ll repeat only the steps that follow.

SUBMIT HOMEWORK: STEP 1

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

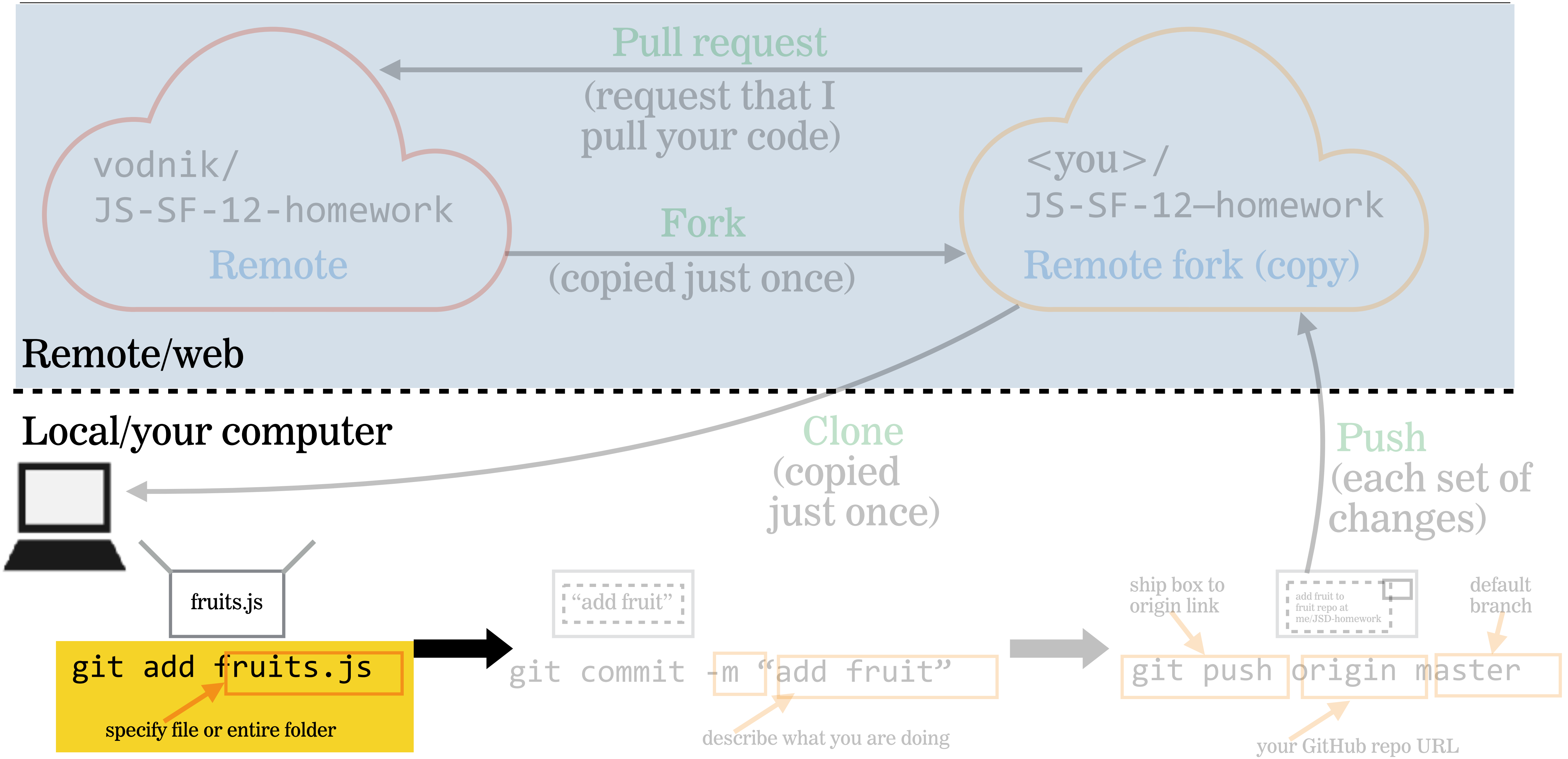


SUBMIT HOMEWORK: STEP 2

In Terminal:

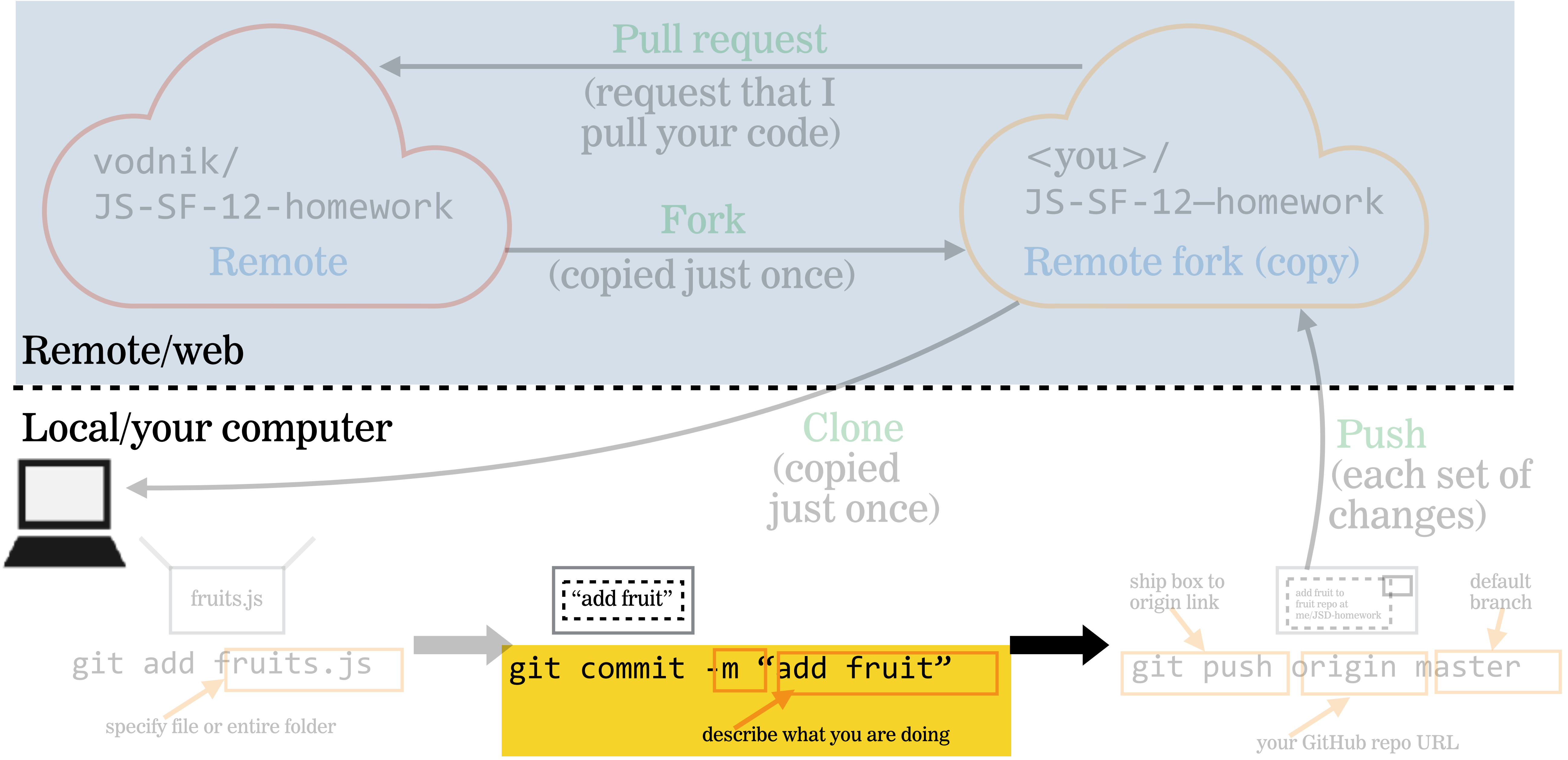
- navigate to `JS-SF-12-homework` folder
- `git add .`
- `git commit -m “submitting Homework 1”`
- `git push origin master`

USING THE JS-SF-12-HOMEWORK REPO



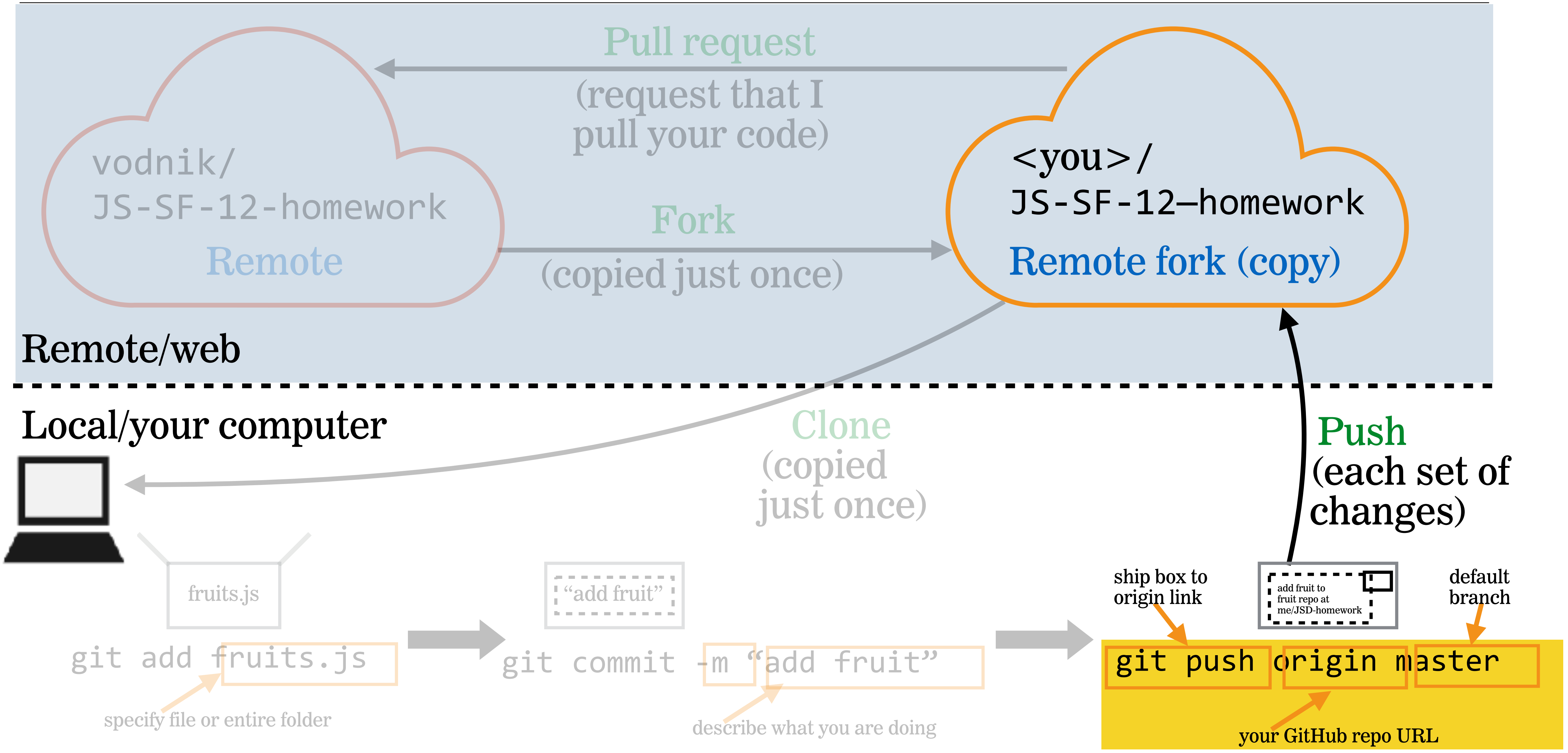
```
git add fruits.js
git commit -m "add fruit"
git push origin master
```

USING THE JS-SF-12-HOMEWORK REPO



```
git add fruits.js
git commit -m “add fruit”
git push origin master
```

USING THE JS-SF-12-HOMEWORK REPO

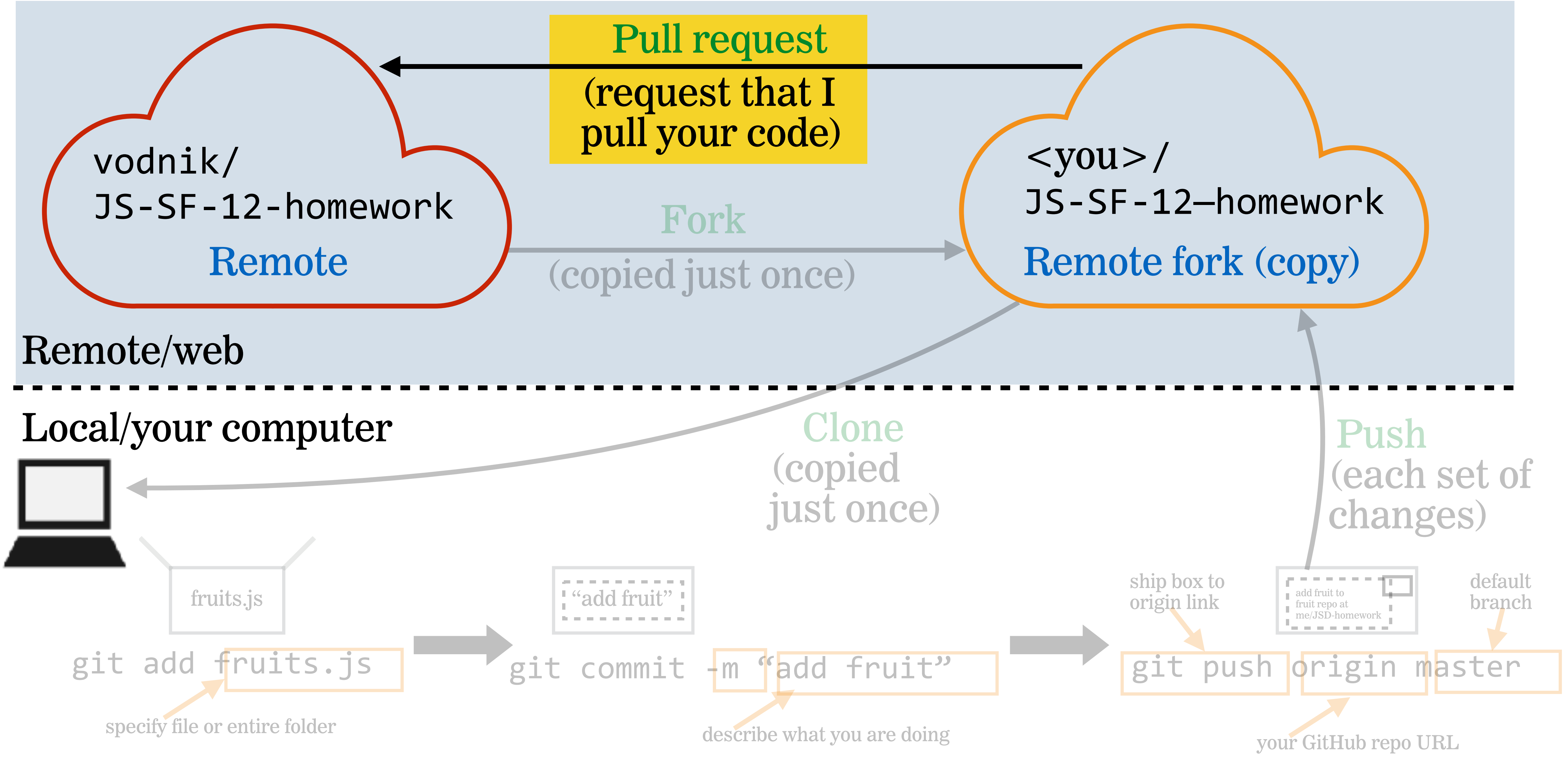


SUBMIT HOMEWORK: STEP 3

In Browser:

- Go to your fork of `JS-SF-12-homework` on `git.generalassemb.ly`
- click **New pull request**
- click **Create pull request**
- click **Create pull request** (again)

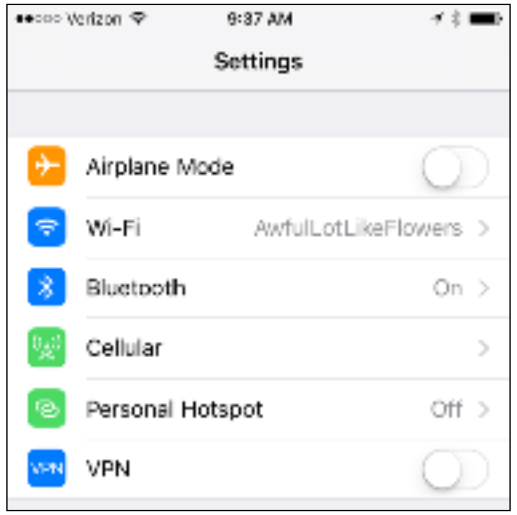
USING THE JS-SF-12-HOMEWORK REPO



```
git add fruits.js
git commit -m "add fruit"
git push origin master
```

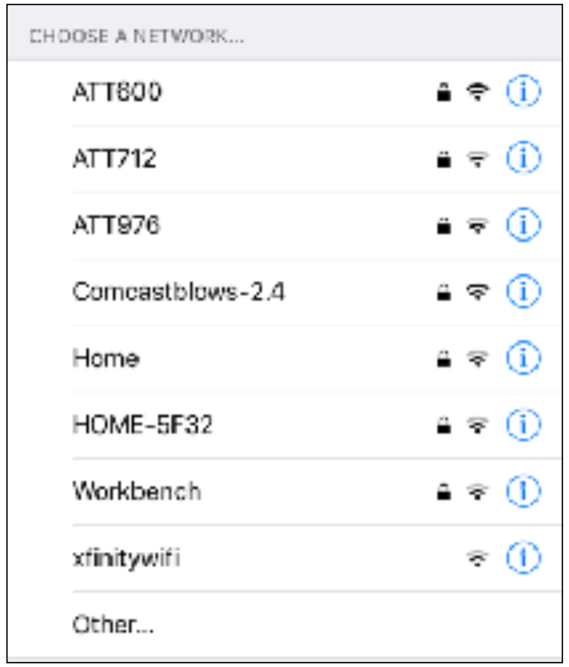
Why do we use different networks to connect to the Internet when we're in different places?

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


●●○○○ Verizon 9:37 AM

Settings

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


CHOOSE A NETWORK...

ATT600
ATT712
ATT976
Comcastblows-2.4
Home
HOME-5F32
Workbench
xfinitywifi
Other...

SCOPE

SCOPE

- Describes the set of variables you have access to

GLOBAL SCOPE

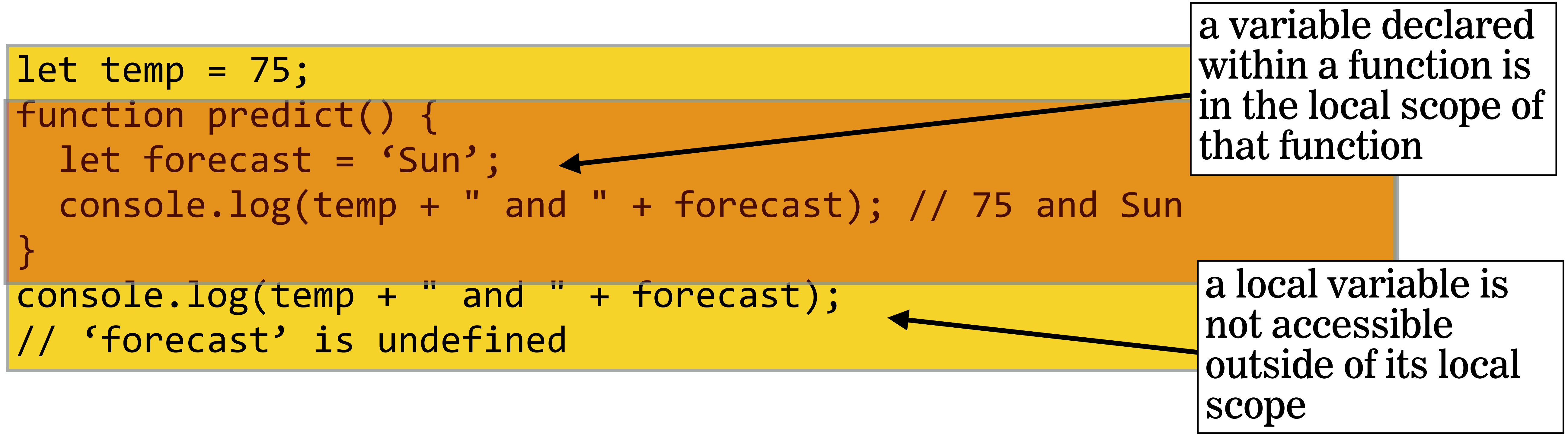
- A variable declared outside of a function is accessible everywhere, even within functions. Such a variable is said to have **global scope**.

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
```

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

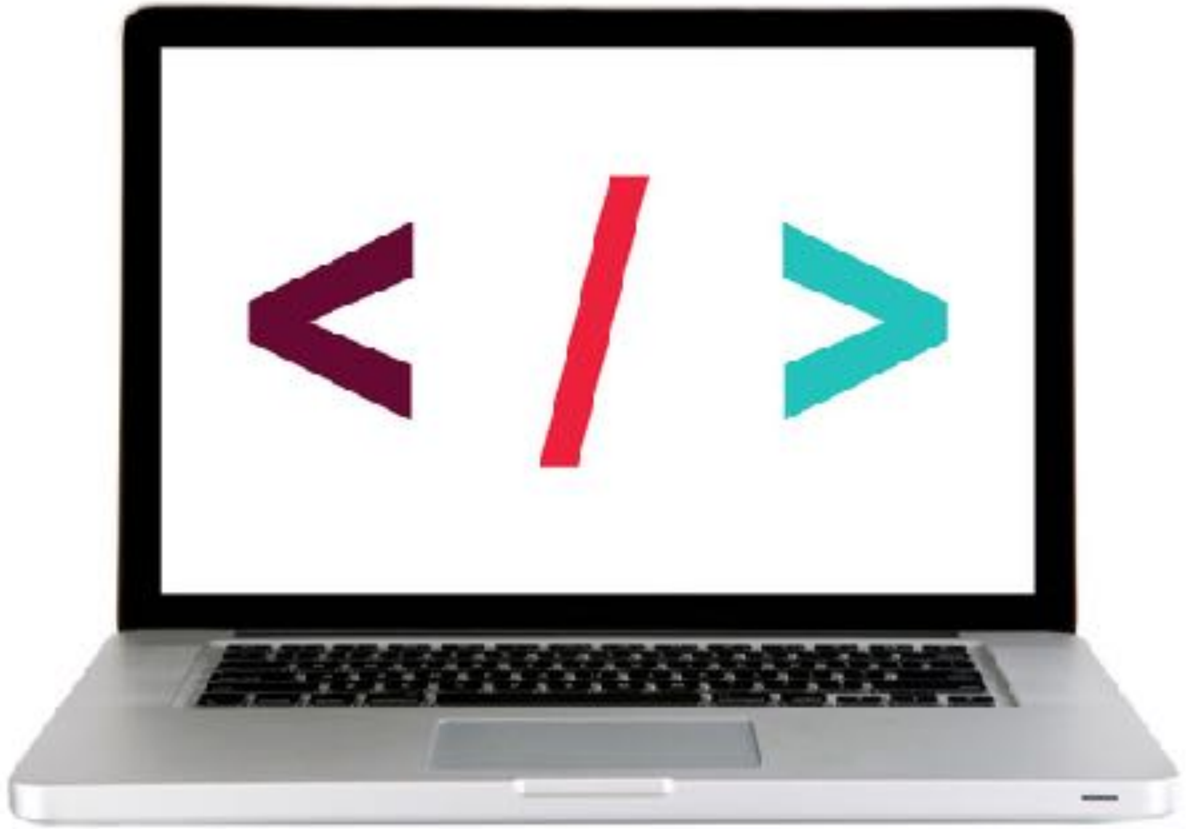
- A variable created with `let` or `const` creates local scope within any block, including blocks that are part of loops and conditionals.
- This is known as **block scope**, which is another type of local scope.

`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

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, and 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.

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

var

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

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

`let`

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

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

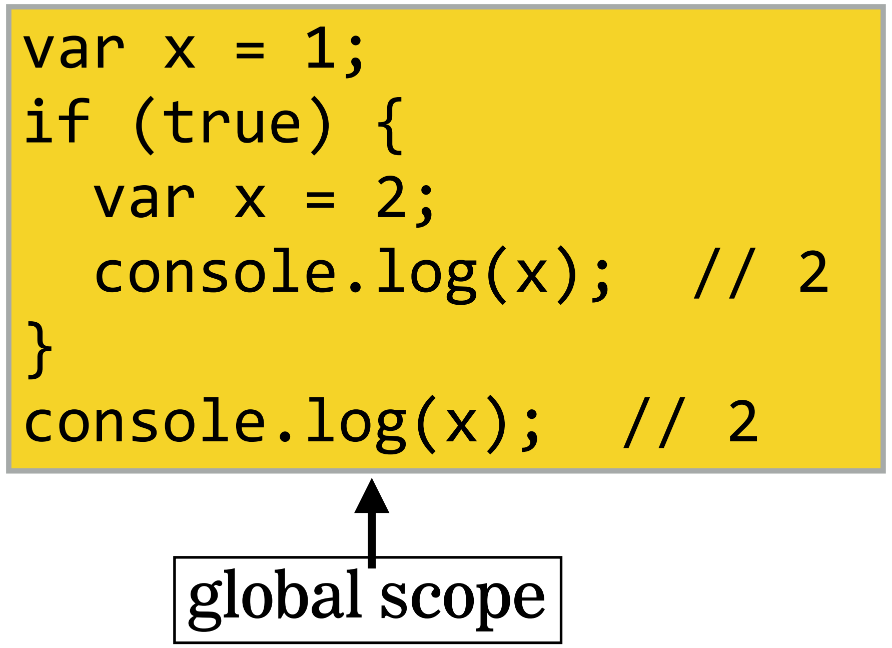
`const`

- `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

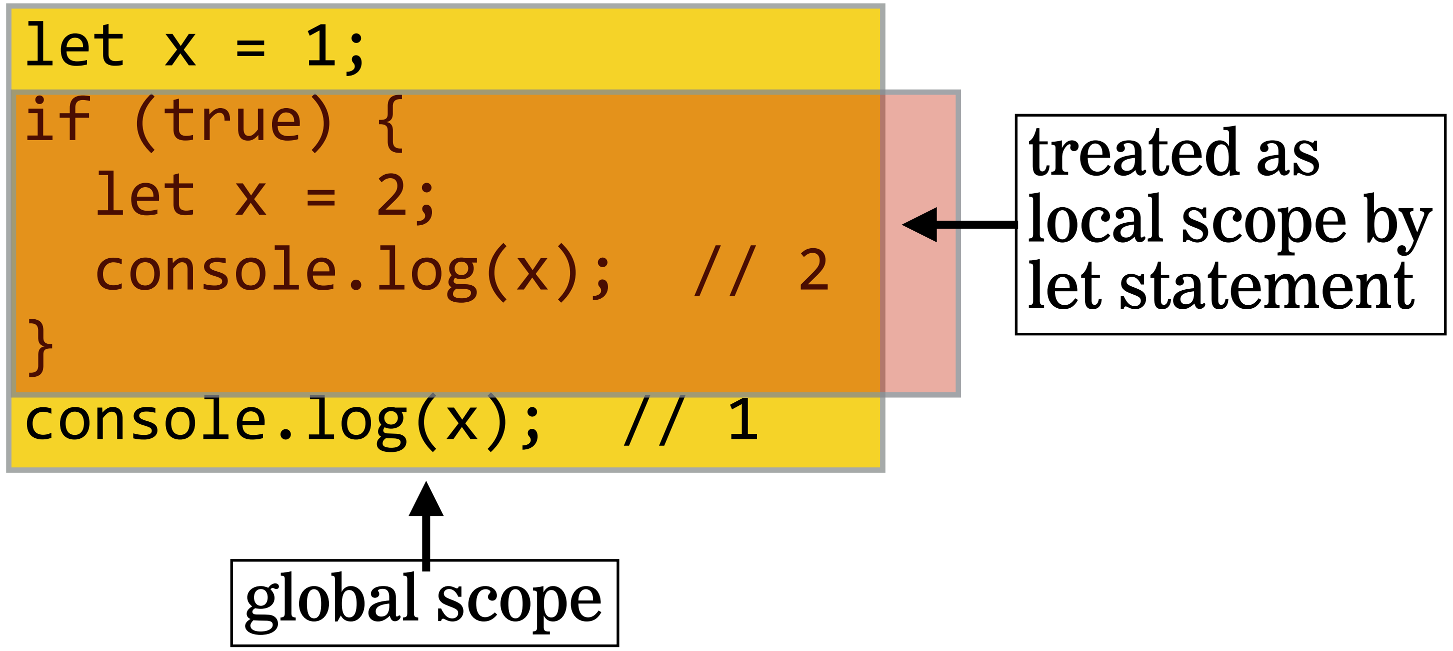
```
const salesTax = 0.0875;
```

`let/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
```

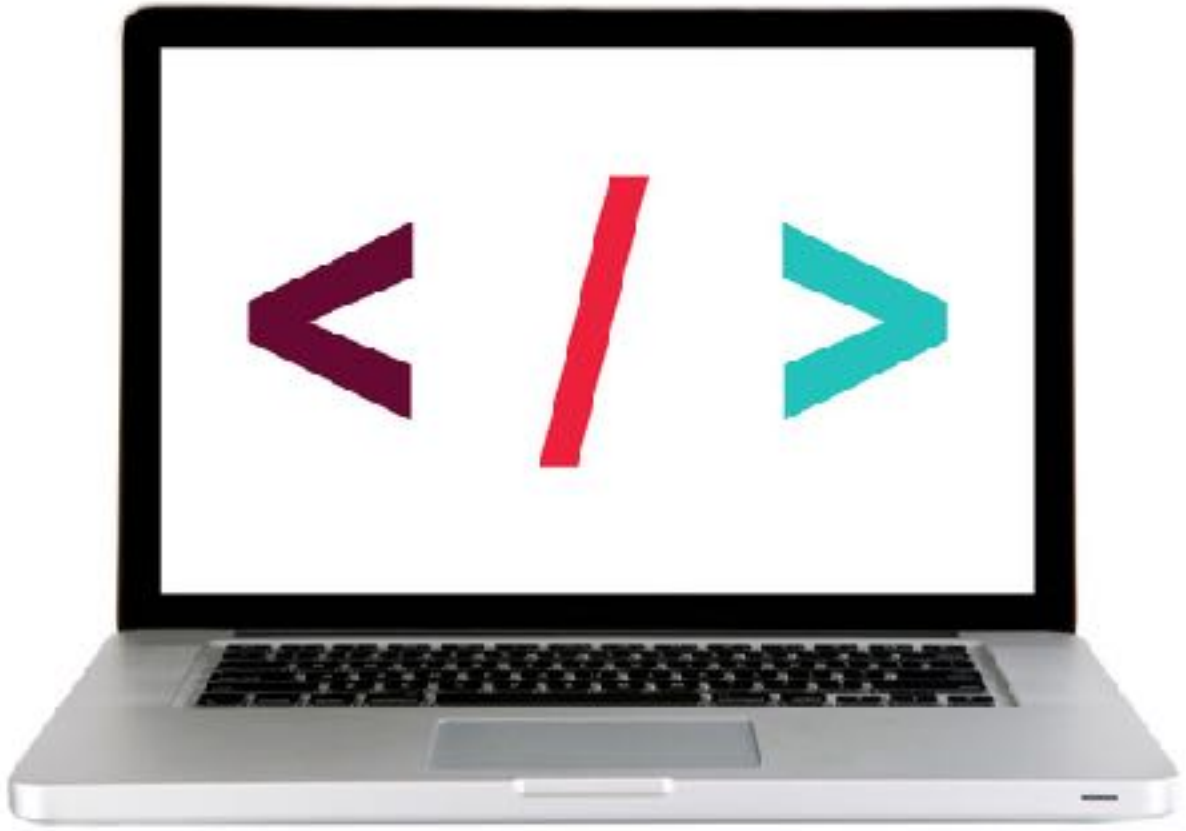
`var`, `let`, `const`, AND BROWSER SUPPORT

- `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
- we will rely on `let` and `const` in class

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

keyword	where does it create local scope?	can you change the value in the current scope?	which browsers support it? (modern or all)
`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**

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

JavaScript moves some declarations to the top of a scope

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: bar is
       // not defined

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

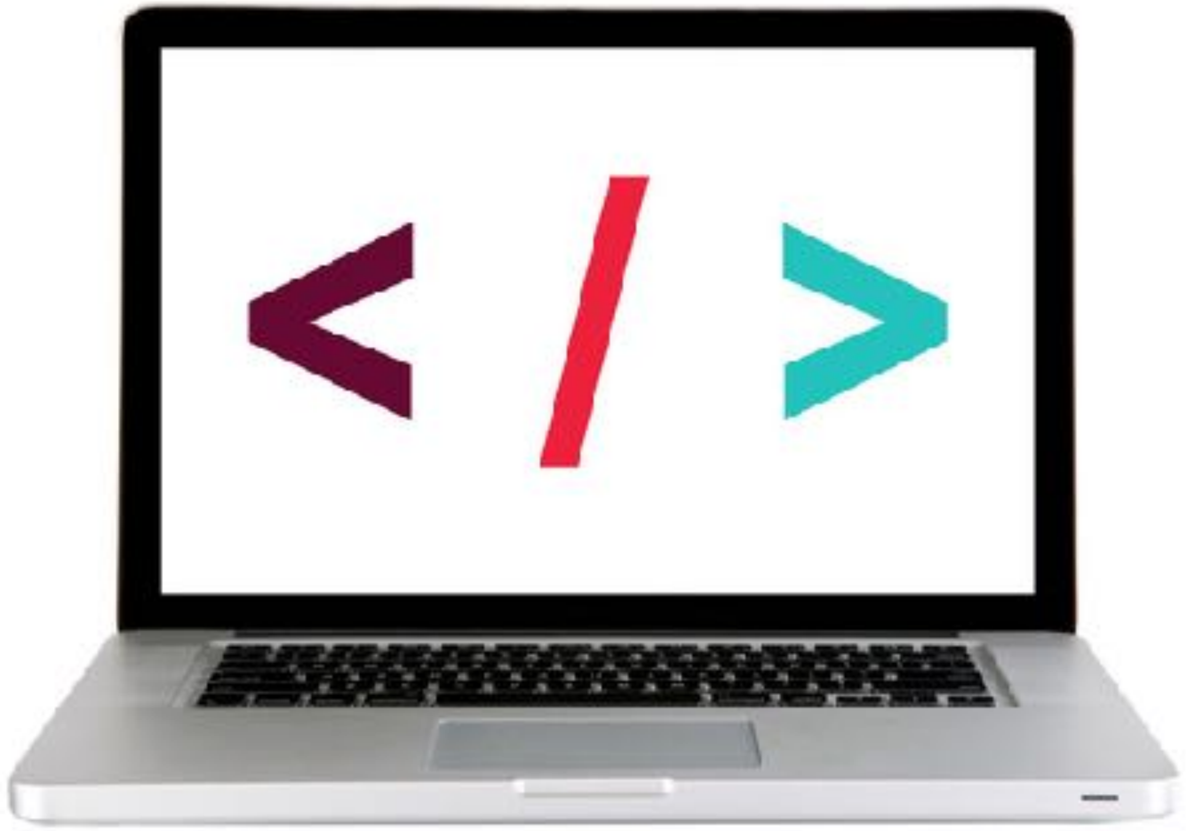
VARIABLES AND HOISTING

keyword	variable name hoisted?	variable value hoisted?
`let`	no	no
`const`	no	no
`var`	yes	no

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.

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

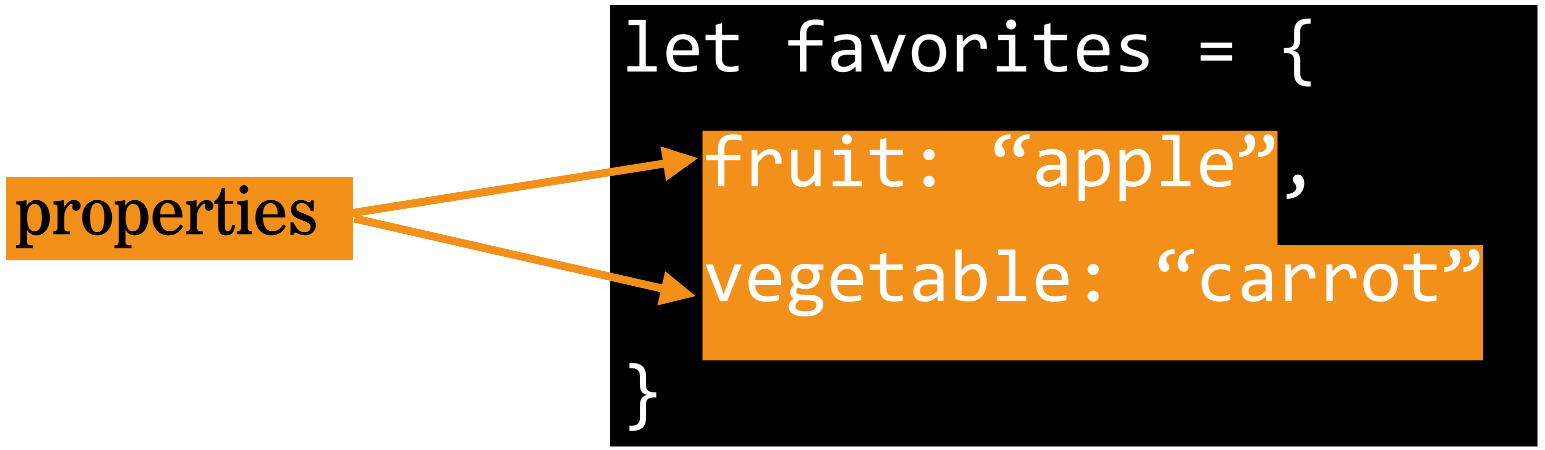
NUMBER

ARRAY

BOOLEAN

OBJECT

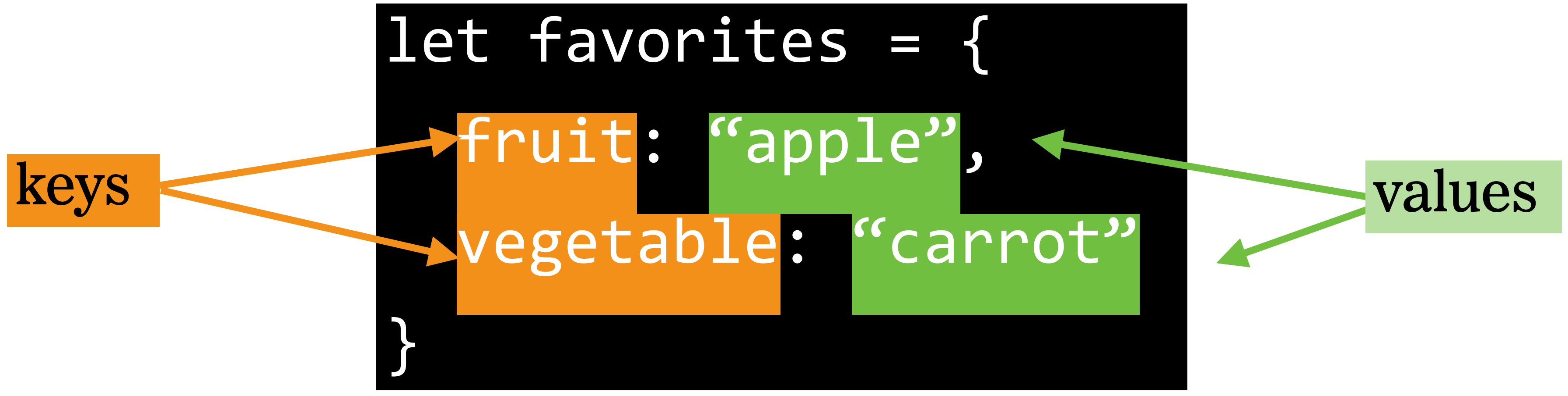
AN OBJECT IS A COLLECTION OF PROPERTIES



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

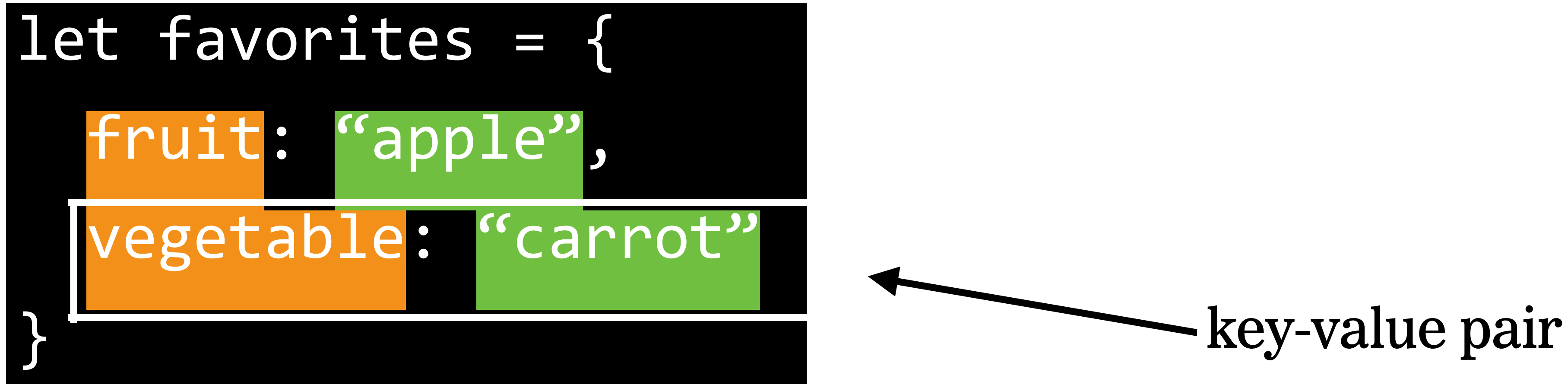
PROPERTY = KEY & VALUE

- 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


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

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!”);
  }
}
```

method

TWO WAYS TO GET/SET PROPERTIES

dot notation

square bracket notation

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”);
  }
}
```

object name

getting properties

```
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”);
  }
}
```

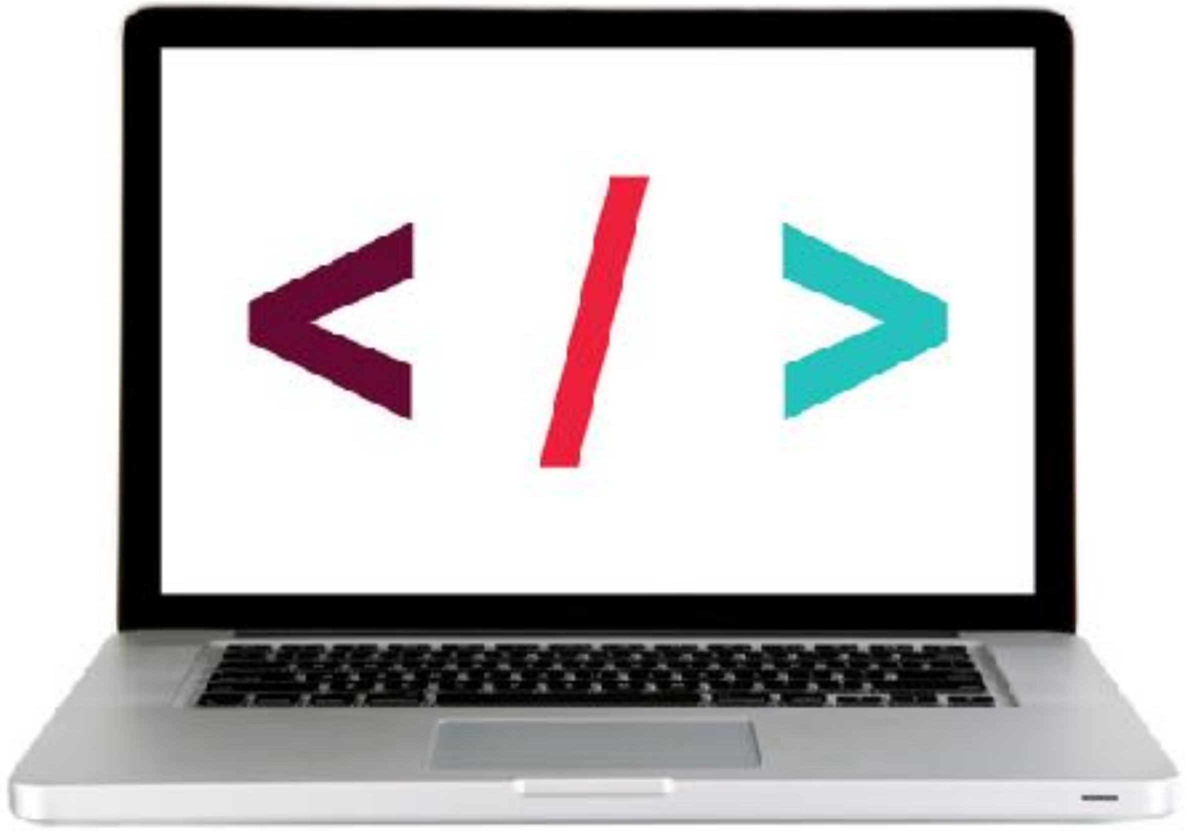
setting properties

```
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

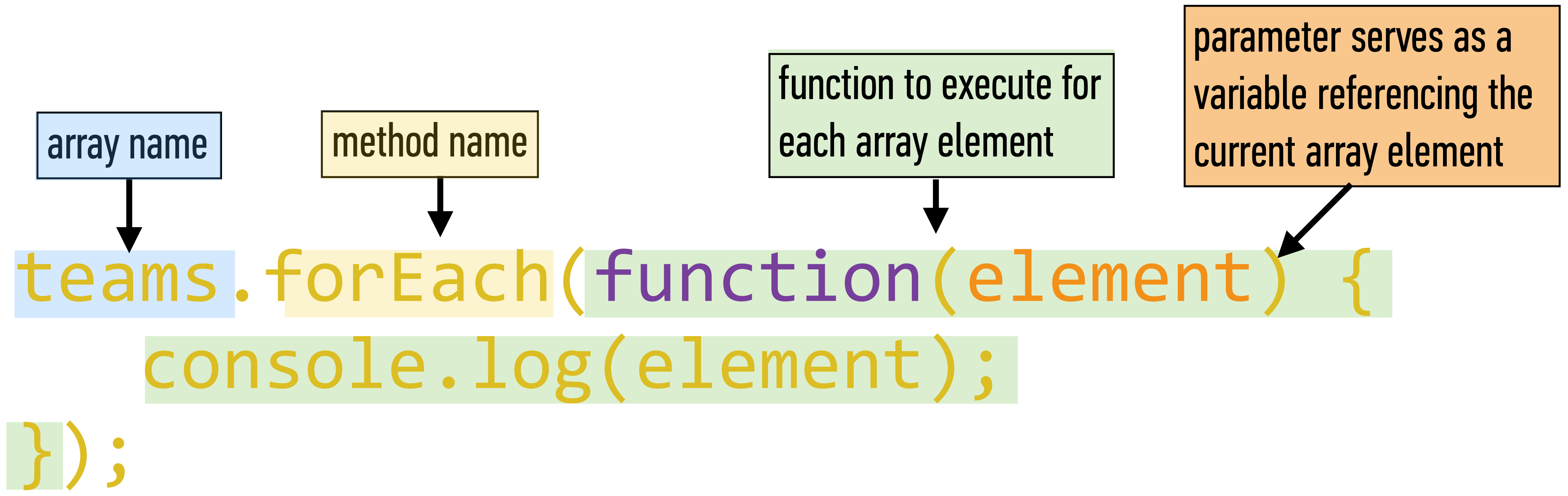
1. On your desk or on the wall, write code to create a variable whose name corresponds to the thing you were assigned in the previous exercise (cloud, houseplant, nation, office chair, or airplane).
2. Write code to add a property to the object and specify a value for the property.
3. Write code to add 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()


array name
method name
function to execute for each array element
parameter serves as a variable referencing the current array element

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

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:	for the shopping cart:
price	contents
color	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

LAB

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.

JSON IS A DATA FORMAT BASED ON JAVASCRIPT

object

```
let instructor = {
  firstName: ‘Sasha’,
  lastName: ‘Vodnik’,
  city: ‘San Francisco’,
  classes: [
    ‘JSD’, ‘FEWD’
  ],
  classroom: 7,
  launched: true,
  dates: {
    start: 20180205,
    end: 20180406
  },
};
```

JSON

```
{
  “firstName”: “Sasha”,
  “lastName”: “Vodnik”,
  “city”: “San Francisco”,
  “classes”: [
    “JSD”, “FEWD”
  ],
  “classroom”: 7,
  “launched”: true,
  “dates”: {
    “start”: 20180205,
    “end”: 20180406
  }
}
```

JSON

- Easy for humans to read and write
- Easy for programs to parse and generate

```
{
  “firstName”: “Sasha”,
  “lastName”: “Vodnik”,
  “city”: “San Francisco”,
  “classes”: [
    “JSD”, “FEWD”
  ],
  “classroom”: 7,
  “launched”: true,
  “dates”: {
    “start”: 20180205,
    “end”: 20180406
  }
}
```

JSON IS NOT JAVASCRIPT-SPECIFIC

- Used across the web by programs written in many languages

JSON RULES

- Property names must be double-quoted strings.
- Trailing commas are forbidden.
- Leading zeroes are prohibited.
- In numbers, a decimal point must be followed by at least one digit.
- Most characters are allowed in strings; however, certain characters (such as ', ", \, and newline/tab) must be 'escaped' with a preceding backslash (\ ) in order to be read as characters (as opposed to JSON control code).
- All strings must be double-quoted.
- No comments!

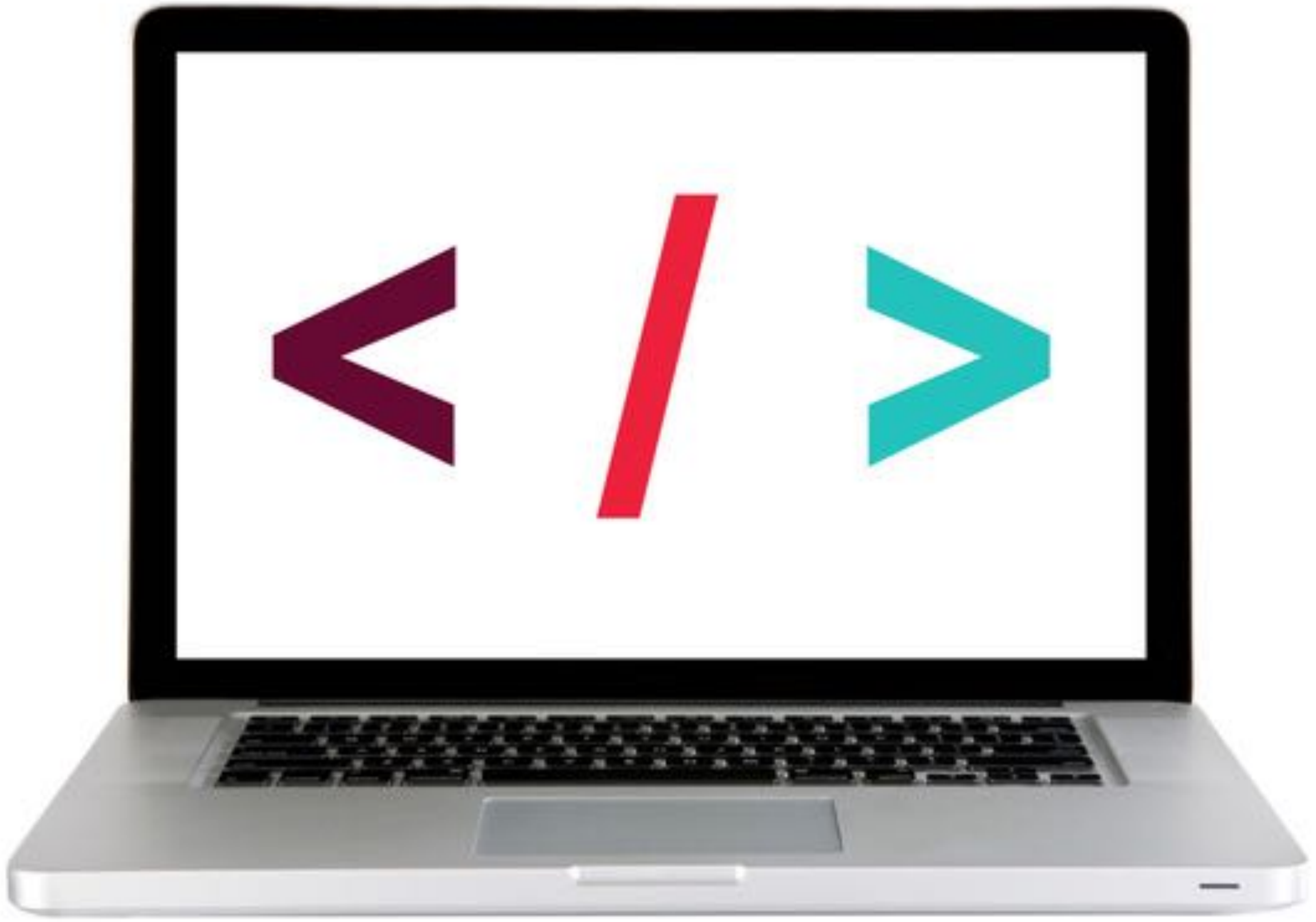
TO CONVERT AN OBJECT TO JSON

```
JSON.stringify(object);
```

TO CONVERT JSON TO AN OBJECT

```
JSON.parse(json);
```

LET'S TAKE A LOOK



EXERCISE — JSON

EXERCISE

KEY OBJECTIVE

- Implement and interface with JSON data

TYPE OF EXERCISE

- Groups of 2-3

TIMING

3 min

1. Write JSON code that contains an error.
2. Write your code on the wall.
3. When everyone's code is done, we will look at the code together as a class and practice identifying errors.

YAY, I GOT SOME DATA!

```
let person = '{"firstName":
"Sasha","lastName": "Vodnik","city":
"San Francisco","classes": ["JSD",
"FEWD"],"classroom": 7,”launched":
true,"dates": {"start": 20180205,"end":
20180406}}’;
```

WAIT, WHAT?!

WORKING WITH NESTED DATA STRUCTURES

1. PARSE THE JSON TO A JAVASCRIPT OBJECT (OR ARRAY!)
2. VIEW THE RESULTING DATA STRUCTURE
3. LOCATE THE DATA YOU WANT TO REFERENCE
4. USE DOT SYNTAX OR SQUARE BRACKET NOTATION TO MOVE DOWN A LEVEL, THEN REPEAT

WORKING WITH NESTED DATA STRUCTURES

1. PARSE THE JSON TO A JAVASCRIPT OBJECT (OR ARRAY!)

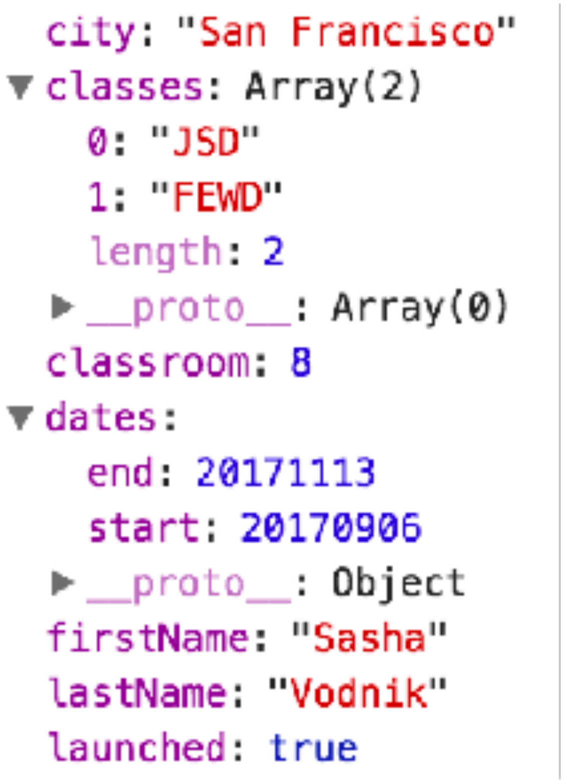
```
let person = '{"firstName":
"Sasha","lastName": "Vodnik","city":
"San Francisco","classes": ["JSD",
"FEWD"],"classroom": 7,”launched":
true,"dates": {"start": 20180205,"end":
20180406}}’;
```

```
let personObject = JSON.parse(person);
```

WORKING WITH NESTED DATA STRUCTURES

2. VIEW THE RESULTING DATA STRUCTURE

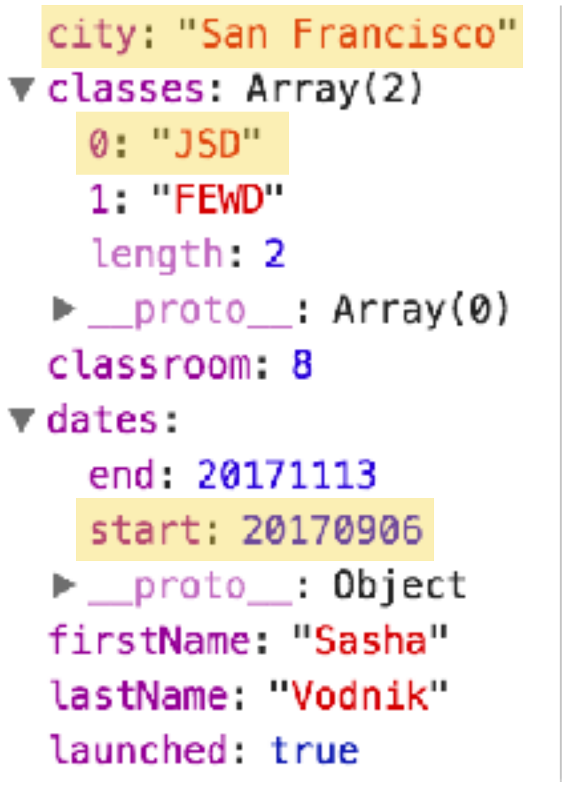
```
let personObject = JSON.parse(person);
console.log(personObject);
>
```



```
  city: "San Francisco"
▼ classes: Array(2)
    0: "JSD"
    1: "FEWD"
    length: 2
  ▶ __proto__: Array(0)
  classroom: 8
▼ dates:
    end: 20171113
    start: 20170906
  ▶ __proto__: Object
  firstName: "Sasha"
  lastName: "Vodnik"
  launched: true
```

WORKING WITH NESTED DATA STRUCTURES

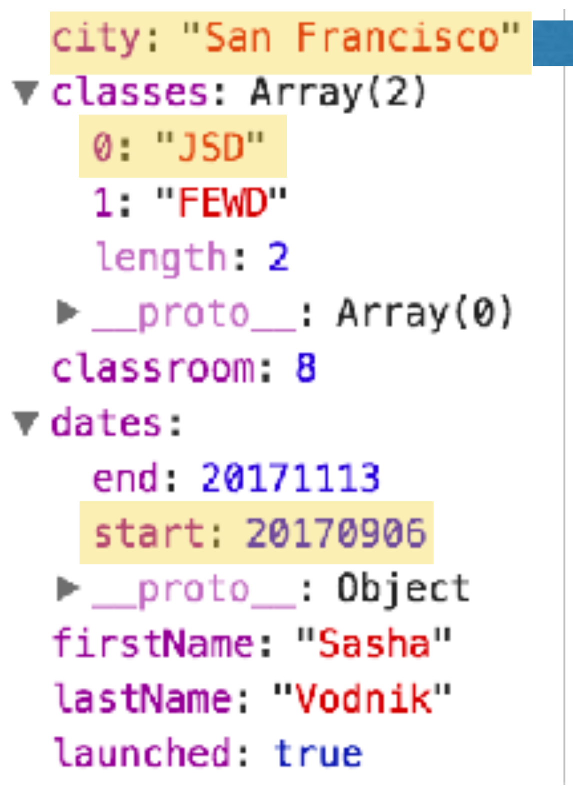
3. LOCATE THE DATA YOU WANT TO REFERENCE



```
  city: "San Francisco"
▼ classes: Array(2)
    0: "JSD"
    1: "FEWD"
    length: 2
  ▶ __proto__: Array(0)
  classroom: 8
▼ dates:
    end: 20171113
    start: 20170906
  ▶ __proto__: Object
  firstName: "Sasha"
  lastName: "Vodnik"
  launched: true
```

WORKING WITH NESTED DATA STRUCTURES

4. USE DOT SYNTAX OR SQUARE BRACKET NOTATION TO MOVE DOWN A LEVEL, THEN REPEAT



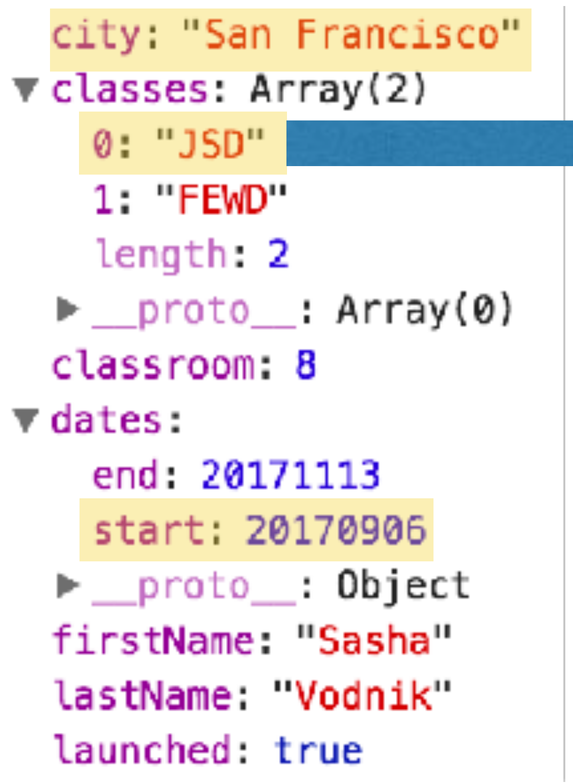
```
city: "San Francisco"
▼classes: Array(2)
  0: "JSD"
  1: "FEWD"
  length: 2
 ▶__proto__: Array(0)
classroom: 8
▼dates:
  end: 20171113
  start: 20170906
 ▶__proto__: Object
firstName: "Sasha"
lastName: "Vodnik"
launched: true
```

direct property:

```
console.log(personObject.city);
> “San Francisco”
```

WORKING WITH NESTED DATA STRUCTURES

4. USE DOT SYNTAX OR SQUARE BRACKET NOTATION TO MOVE DOWN A LEVEL, THEN REPEAT



```
  city: "San Francisco"
▼ classes: Array(2)
    0: "JSD"
    1: "FEWD"
    length: 2
  ▶ __proto__: Array(0)
  classroom: 8
▼ dates:
    end: 20171113
    start: 20170906
  ▶ __proto__: Object
  firstName: "Sasha"
  lastName: "Vodnik"
  launched: true
```

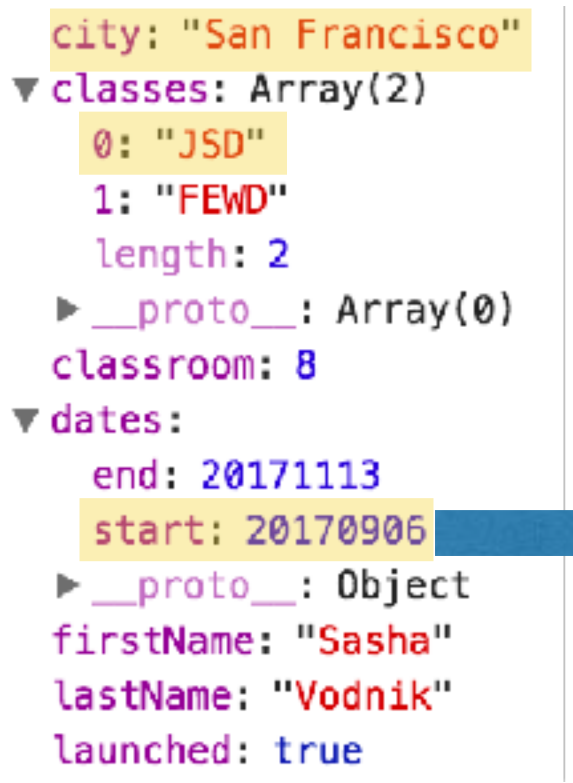
direct property > array element

```
console.log(personObject.classes);
> [“JSD”,”FEWD”]

console.log(personObject.classes[0]);
> “JSD”
```

WORKING WITH NESTED DATA STRUCTURES

4. USE DOT SYNTAX OR SQUARE BRACKET NOTATION TO MOVE DOWN A LEVEL, THEN REPEAT



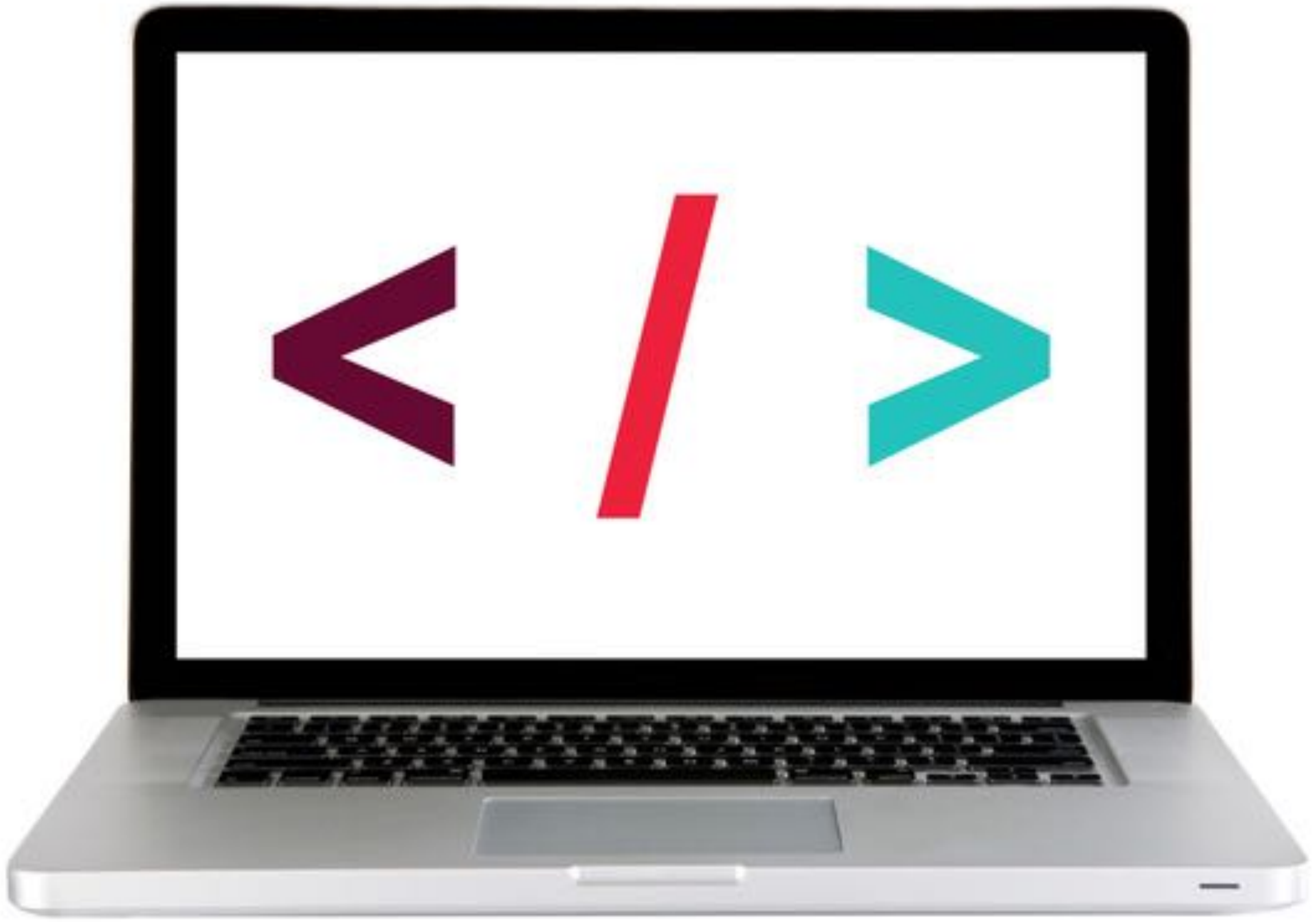
```
  city: "San Francisco"
▼ classes: Array(2)
    0: "JSD"
    1: "FEWD"
    length: 2
  ▶ __proto__: Array(0)
  classroom: 8
▼ dates:
    end: 20171113
    start: 20170906
  ▶ __proto__: Object
  firstName: "Sasha"
  lastName: "Vodnik"
  launched: true
```

direct property > nested object property

```
console.log(personObject.dates);
> {end:20171113,start:20170906}

console.log(personObject.dates.start);
> 20170906
```

LET'S TAKE A LOOK



LAB — JSON

LAB

KEY OBJECTIVE

- Implement and interface with JSON data

TYPE OF EXERCISE

- Individual or pair

TIMING

10 min

1. Open `starter-code > 2-json-exercise > app.js` in your editor.
2. Follow the instructions to write code that produces the stated output.

Exit Tickets!

(Class #4)

LEARNING OBJECTIVES – REVIEW

- Determine the scope of local and global variables
- Create a program that hoists variables
- Identify likely objects, attributes, 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