GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Submit homework and create a pull request
2. Pull changes from the `svodnik/JS-SF-8-resources` repo to your computer
3. Open the `starter-code` folder in your code editor

ASYNCHRONOUS JAVASCRIPT AND CALLBACKS

LEARNING OBJECTIVES

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

- Implement a jQuery Ajax client for a simple REST service.
- Pass functions as arguments to functions that expect them.
- Write functions that take other functions as arguments.
- Build asynchronous program flow using promises and Fetch

AGENDA

- Implement a jQuery Ajax client for a simple REST service.
- Functions as callbacks
- Promises & Fetch

WEEKLY OVERVIEW

WEEK 7	Asynchronous JavaScript & Callbacks / Advanced APIs
WEEK 8	Project 2 Lab / Context and `this`
WEEK 9	CRUD & Firebase / Deploying your app

HOMEWORK REVIEW

HOMEWORK — GROUP DISCUSSION

EXERCISE

TYPE OF EXERCISE

- Groups of 3

TIMING

4 min

1. Share your solutions for the homework.
2. Share a challenge you encountered, and how you overcame it.
3. Share 1 thing you found challenging. If you worked it out, share how; if not, brainstorm with your group how you might approach it.
4. Share the APIs you plan to use for the Feedr project, and what you've learned about them from their documentation.

EXIT TICKET QUESTIONS

1. > Headers? I've read a lot about headers, but don't know exactly how to use...for example in CORS, putting the content-type on a request, etc
   > More examples of HTTP Request format?
2. I'm just not sure I understand how APIs and AJAX and REST are all connected. I feel confident writing the code, but don't really know how to explain what it's doing. A lot of this still feels pretty nebulous.
3. Do you link all of your JS files to your main.js if you want to keep them separate?

EXIT TICKET QUESTIONS

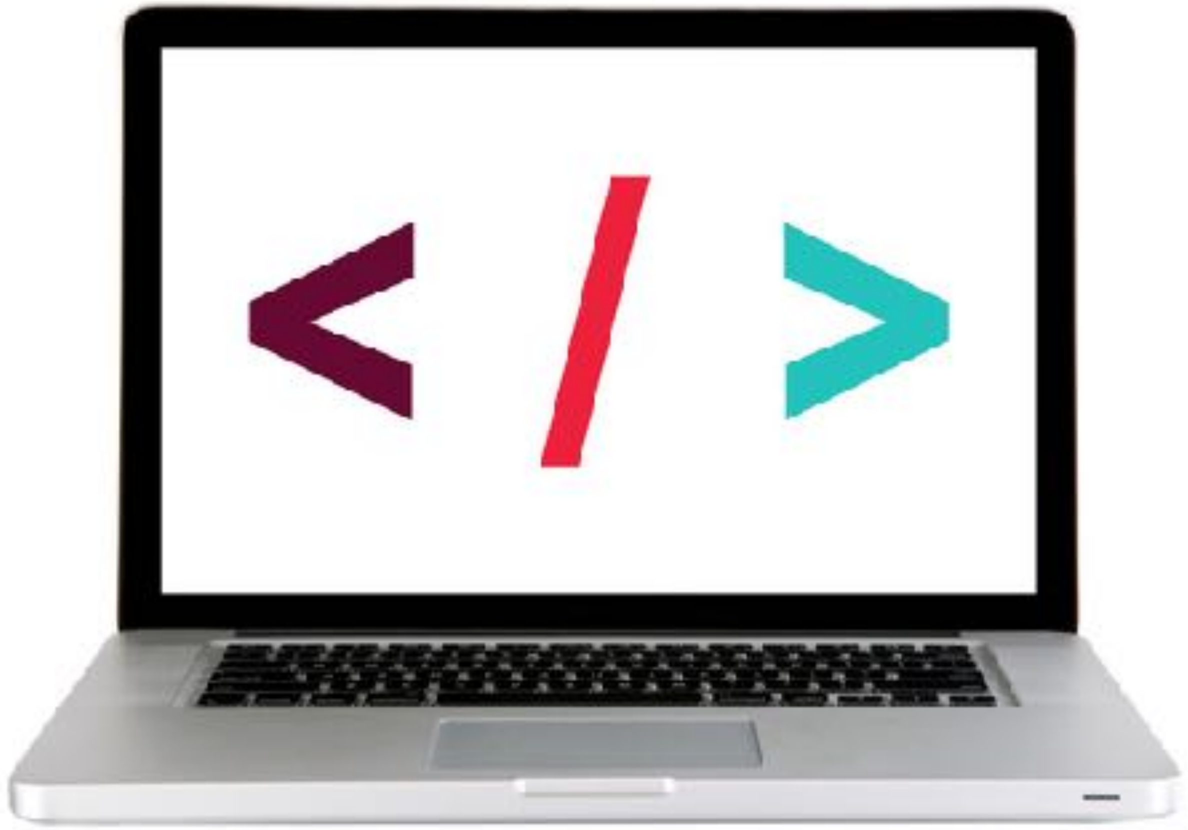
4. How to find interesting APIs and how to navigate them and find the right information. How to get better at following the .next().next().next() workflow (vs. line-by-line code)
5. The data we got back from our fetch requests came in different formats. One was an array and one was a javascript object. Why do we get different data types? Is that just how that API stores data?

jQuery Ajax

Using Ajax with jQuery

method	description
`$.get()`	loads data from a server using an HTTP GET request
`$.ajax()`	performs an Ajax request based on parameters you specify

LET'S TAKE A CLOSER LOOK



LAB — JQUERY AJAX

OBJECTIVES

- Implement a jQuery Ajax client for a simple REST service.

LOCATION

- `starter-code > 1-jquery-ajax-exercise`

EXECUTION

10 min

1. Read the documentation at zippopotam.us. Note that zippopotam.us does not require an API key.
2. Create an ajax request using the jQuery get() method. Log the response to the console.
3. Write code to identify that city and state values in the response and the update your code to log only those values to the console.
4. Bonus items are detailed in the script.js file.

ACTIVITY

EXERCISE

TYPE OF EXERCISE

- Turn & Talk

TIMING

2 min

1. For the code on the board, identify the number of arguments.
2. Find a partner or two, share your answers, and discuss.

HOW MANY ARGUMENTS IN THIS CODE?

```
button.addEventListener('click', function() {
  // your code here
}, false);
```

Functions and callbacks

SYNCHRONOUS PROGRAMMING

```
function doSomething() {
    // do something
}
function doAnotherThing() {
    // do another thing
}
function doSomethingElse() {
    // do one more thing
}
```

run each function, one after the other

```
doSomething();
doAnotherThing();
doSomethingElse();
```

ASYNCHRONOUS PROGRAMMING

```
function doSomething() {
    // do something
}
function doAnotherThing() {
    // do another thing
}
function doSomethingElse() {
    // do one more thing
}
```

run each function, but only after something has happened

```
$(‘button’).on(‘click’, doSomething);
```

```
$.get(url, function(data) {
  doAnotherThing(data);
});
```

```
fetch(url).then(function(response) {
  if (response.ok) {
    return response.json();
  } else {
   console.log('There was a problem.');
  }
}).then(doSomethingElse(data));
```

FUNCTIONS ARE FIRST-CLASS OBJECTS

- Functions can be used in any part of the code that strings, arrays, or data of any other type can be used
  - ➡store functions as variables
  - ➡pass functions as arguments to other functions
  - ➡return functions from other functions
  - ➡run functions without otherwise assigning them

HIGHER-ORDER FUNCTION

- A function that takes another function as an argument, or that returns a function

HIGHER-ORDER FUNCTION — EXAMPLE

`setTimeout()`

```
setTimeout(function, delay);
```

where

- `function` is a function (reference or anonymous)
- `delay` is a time in milliseconds to wait before the first argument is called

SETTIMEOUT WITH ANONYMOUS FUNCTION ARGUMENT

```
setTimeout(function(){
  console.log("Hello world");
}, 1000);
```

SETTIMEOUT WITH NAMED FUNCTION ARGUMENT

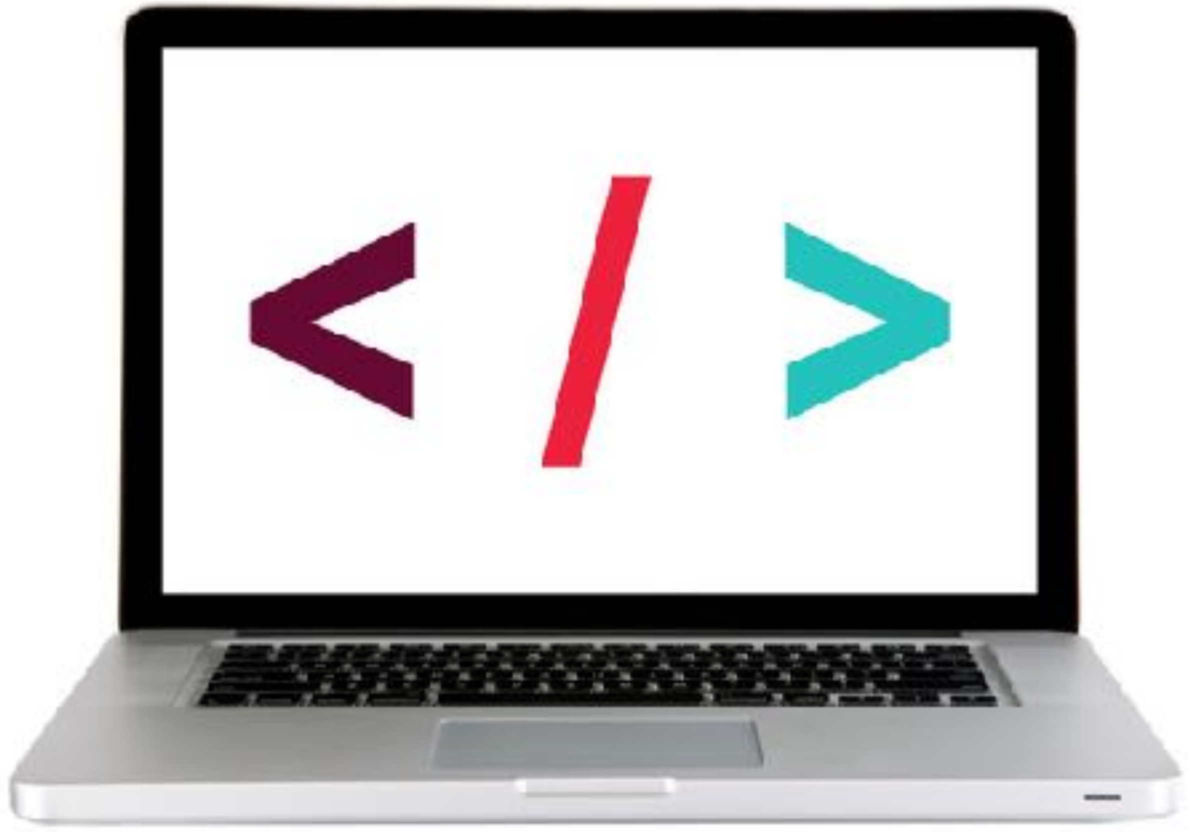
```
function helloWorld() {
  console.log("Hello world");
}

setTimeout(helloWorld, 1000);
```

CALLBACK

- A function that is passed to another function as an argument, and that is then called from within the other function
- A callback function can be anonymous (as with `setTimeout()` or `forEach()`) or it can be a reference to a function defined elsewhere

LET'S TAKE A CLOSER LOOK



EXERCISE - CREATING A CALLBACK FUNCTION, PART 1

EXERCISE

LOCATION

```
‣ starter-code > 3-callback-exercise
```

TIMING

10 min

1. In your editor, open script.js.
2. Follow the instructions in Part 1 to create the add, process, and subtract functions, and to call the process function using the add and subtraction functions as callbacks.
3. Test your work in the browser and verify that you get the expected results.
4. BONUS: Comment out your work and recreate using arrow functions (see https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions)

EXERCISE – CREATING A CALLBACK FUNCTION, PART 2

EXERCISE

LOCATION

▸ `starter-code > 3-callback-exercise`

TIMING

10 min

1. In your editor, return to script.js.
2. Follow the instructions in Part 2 to allow the process function to accept values as additional parameters, and to pass those values when calling the callback function.
3. Test your work in the browser and verify that you get the expected results.
4. BONUS: Make the same changes to your code that uses arrow functions.

Promises & Fetch

PROMISES

traditional callback:

```
doSomething(successCallback, failureCallback);
```

callback using a promise:

```
doSomething().then(successCallback, failureCallback);
```

MULTIPLE CALLBACKS — TRADITIONAL CODE

```
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log('Got the final result: ' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

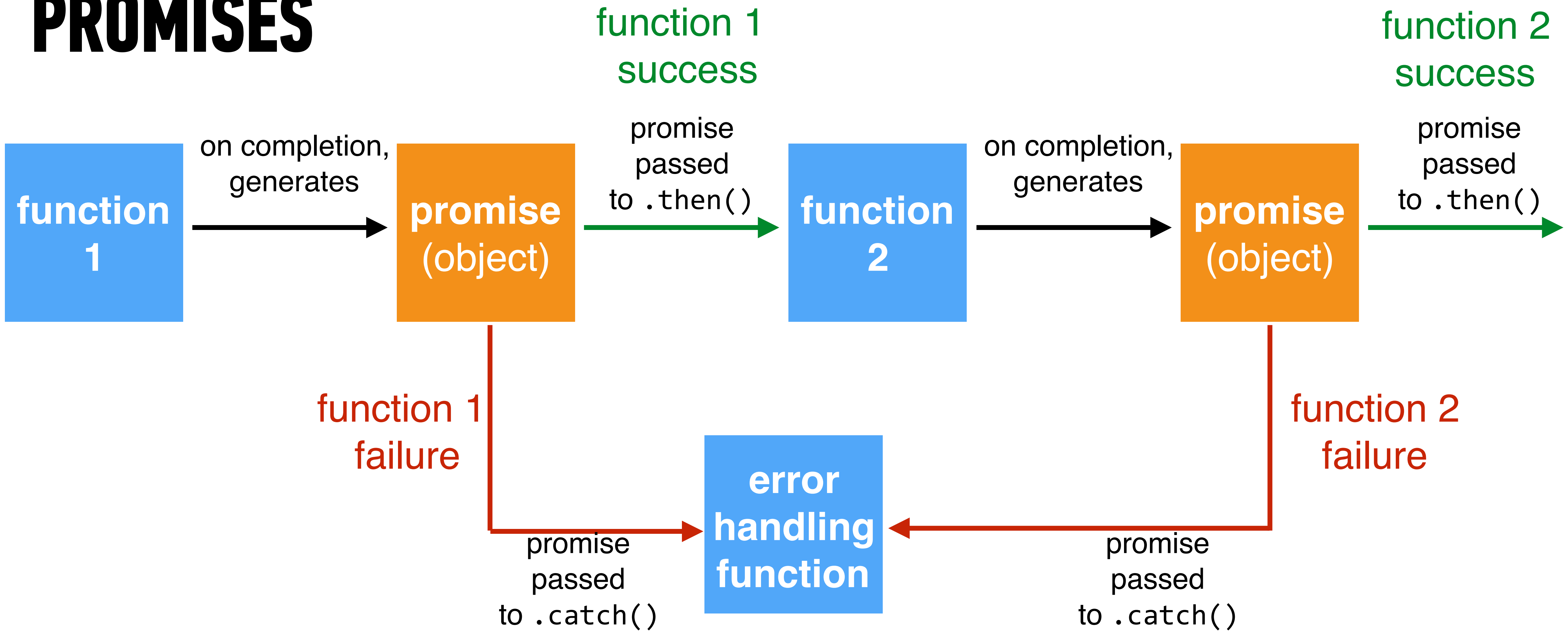
MULTIPLE CALLBACKS WITH PROMISES

```
doSomething().then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```

ERROR HANDLING WITH PROMISES

```
doSomething().then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```

PROMISES


function 1
success
function 2
success
function
1
on completion, generates
promise
(object)
promise passed to .then()
function
2
on completion, generates
promise
(object)
promise passed to .then()
function 1
failure
function 2
failure
error handling function
promise passed to .catch()
promise passed to .catch()

FETCH

```
fetch(url).then(function(response) {
  if(response.ok) {
    return response.json();
  }
  throw new Error('Network response was not ok.');
}).then(function(data) {
  // DOM manipulation
}).catch(function(data) {
  // handle lack of data in UI
});
```

FETCH VS JQUERY $.GET()

```
fetch(url).then(function(res) {
  if(response.ok) {
    return res.json();
  }
  throw new Error(‘problem');
}).then(function(data) {
  // DOM manipulation
}).catch(function(data) {
  // handle lack of data in UI
});
```

```
$.get(url).done(function(data) {
  // work with data
}).done(function(data) {
  // DOM manipulation
})
.fail(function(data) {
  // handle lack of data in UI
});
```

ERROR HANDLING FOR INITIAL FETCH REQUEST

```
fetch(url).then(function(response) {
  if(response.ok) {
    return response.json();
  }
  throw new Error('Network response was not ok.');
}).then(function(data) {
  // DOM manipulation
}).catch(function(data) {
  // handle lack of data in UI
});
```

Exit Tickets!

(Class #11)

LEARNING OBJECTIVES – REVIEW

- Implement a jQuery Ajax client for a simple REST service.
- Pass functions as arguments to functions that expect them.
- Write functions that take other functions as arguments.
- Build asynchronous program flow using promises and Fetch

NEXT CLASS PREVIEW

Advanced APIs

- Generate API specific events and request data from a web service.
- Implement a geolocation API to request a location.
- Process a third-party API response and share location data on your website.
- Make a request and ask another program or script to do something.
- Search documentation needed to make and customize third-party API requests.

Q&A