GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Remove your API keys from your homework, then submit it and create a pull request
2. Pull changes from the `svodnik/JS-SF-9-resources` repo to your computer
3. Open the `11-async-callbacks > starter-code` folder in your code editor

JAVASCRIPT DEVELOPMENT

ASYNCHRONOUS JAVASCRIPT & CALLBACKS

LEARNING OBJECTIVES

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

- 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

- Functions as callbacks
- Promises & Fetch

WEEKLY OVERVIEW

WEEK 7	Asynchronous JavaScript & Callbacks / Advanced APIs
	HOLIDAY WEEK — NO CLASS!
WEEK 8	Project 2 Lab / Closures & the module pattern
WEEK 9	CRUD & Firebase / Deploying your app

HOMEWORK REVIEW

HOMEWORK — GROUP DISCUSSION

EXERCISE

TYPE OF EXERCISE

- Groups of 3

TIMING

6 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. How much freedom do we have to modify design of api data?
2. Are there compliers to convert JQuery to Javascript?

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);
```

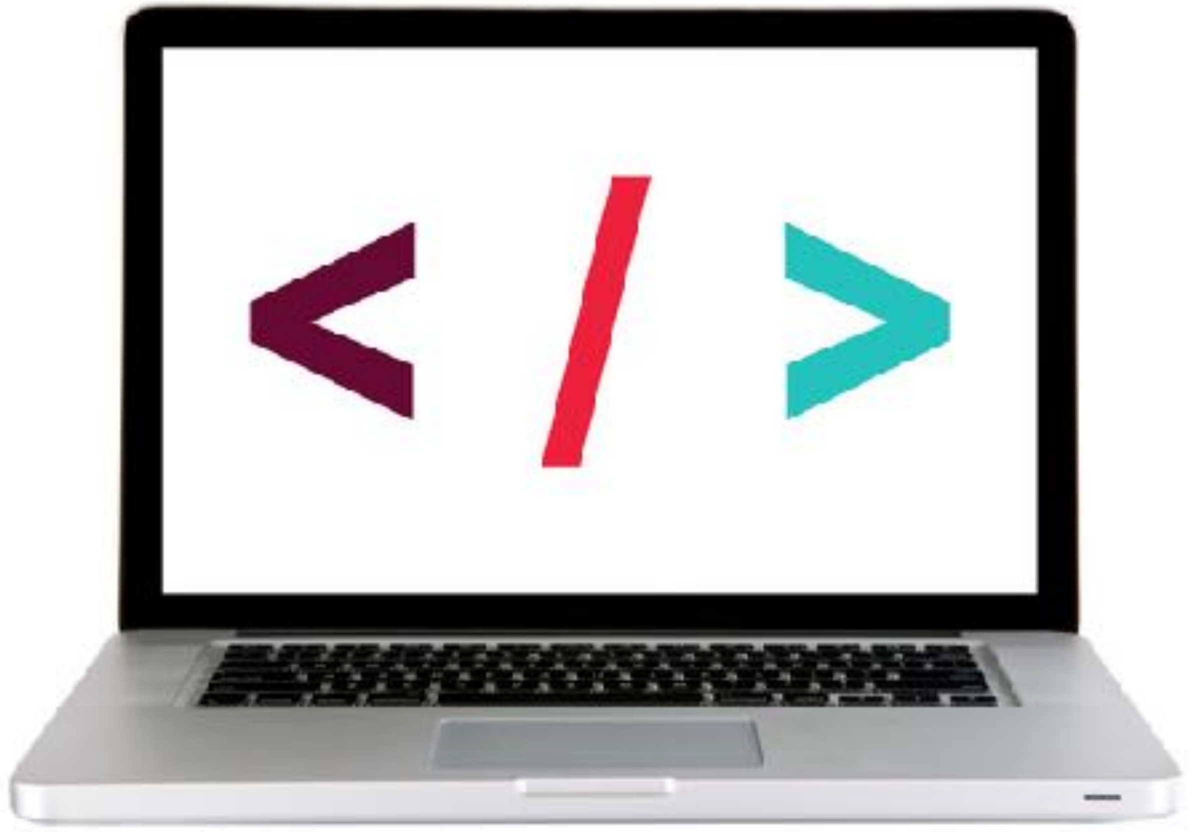
SETTIMEOUT WITH NAMED FUNCTION ARGUMENT

```
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(
  // work with result
).catch(
  // handle error
);
```

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);
```

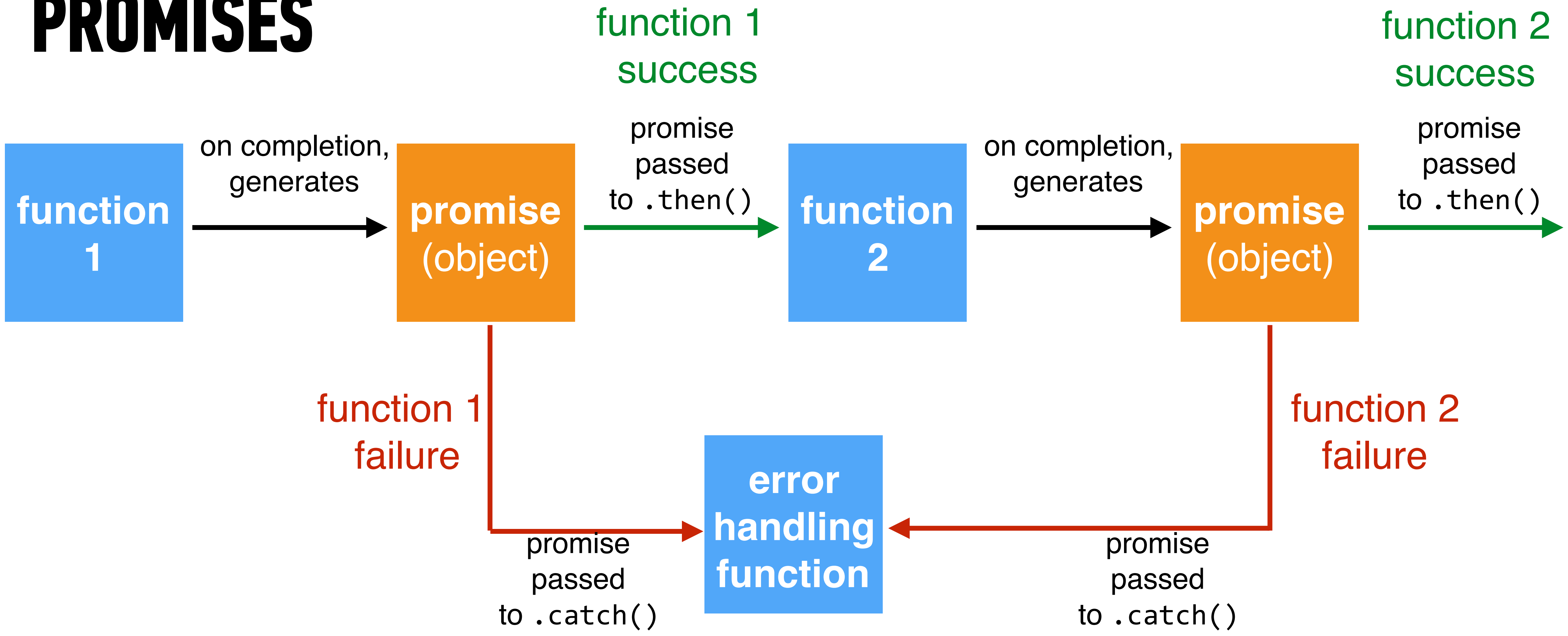
MULTIPLE CALLBACKS WITH PROMISES

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

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(function(error) {
  console.log('There was an error');
});
```

PROMISES


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

FETCH

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

ASYNCHRONOUS JAVASCRIPT & CALLBACKS

Fetch

```
fetch(url).then(function(res) {
  if(res.ok) {
    return res.json();
  } else {
    throw ‘problem';
  }
}).then(function(data) {
  // DOM manipulation

}).catch(function(error) {
  // handle lack of data in UI
});
```

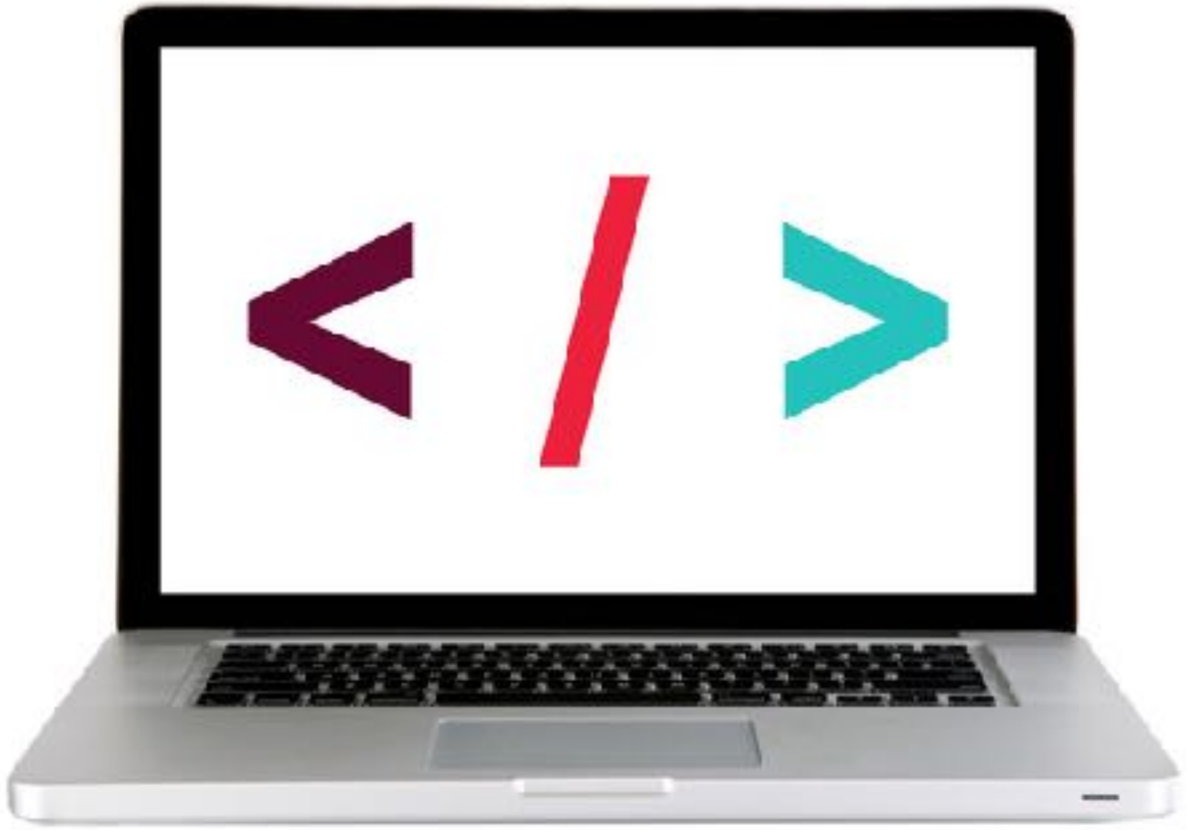
jQuery .get()

```
$.get(url).done(function(data) {
  // DOM manipulation
})







.fail(function(error) {
  // handle lack of data in UI
});
```

ERROR HANDLING FOR INITIAL FETCH REQUEST

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

LET'S TAKE A CLOSER LOOK



EXERCISE - FETCH

EXERCISE

LOCATION

▸ `starter-code > 3-async-exercise`

TIMING

until 9:20

1. In your editor, open script.js.
2. Follow the instructions to add a Fetch request for weather data that uses the results of the existing zip code lookup.

Exit Tickets!

(Class #11)

LEARNING OBJECTIVES - REVIEW

- 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