GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `svodnik/JS-SF-15-resources` repo to your computer
2. Open the `09-ajax-apis/starter-code` folder in your code editor

AJAX & ASYNCHRONOUS JAVASCRIPT

LEARNING OBJECTIVES

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

- Access public APIs and get information back.
- Implement an Ajax request with Fetch.
- Create an Ajax request using jQuery.
- Describe what asynchronous means in relation to JavaScript
- Pass functions as arguments to functions that expect them.
- Write functions that take other functions as arguments.
- Build asynchronous program flow using Fetch

AGENDA

- Ajax using Fetch
- Ajax & jQuery
- Separation of concerns
- Asynchronous code
- Functions as callbacks
- Promises & Fetch

WEEKLY OVERVIEW

WEEK 5	Advanced jQuery & APIs / Ajax & asynchronous JS
WEEK 6	Asynchronous JS & callbacks / Advanced APIs
WEEK 7	Project 2 lab / Prototypal inheritance

EXIT TICKET QUESTIONS

AJAX & ASYNC

Ajax

Ajax

A synchronous

J avaScript

A nd

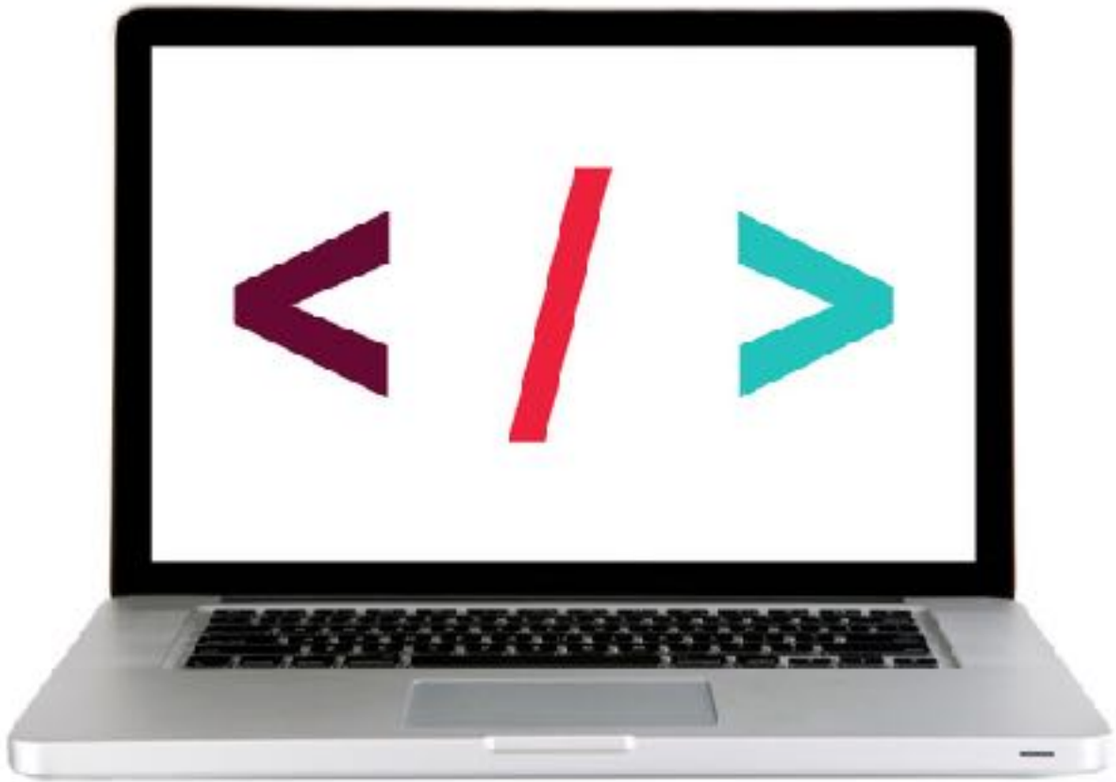
X ML **or JSON!**

Ajax in vanilla JS

Fetch = Ajax requests in vanilla JavaScript

```
fetch(url)
  .then((response) => {
    // check if request was successful
  })
  .then((data) => {
    // do something with the data
  });
```

LET'S TAKE A CLOSER LOOK



EXERCISE - CREATING AN AJAX REQUEST

LOCATION

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

TIMING

5 min

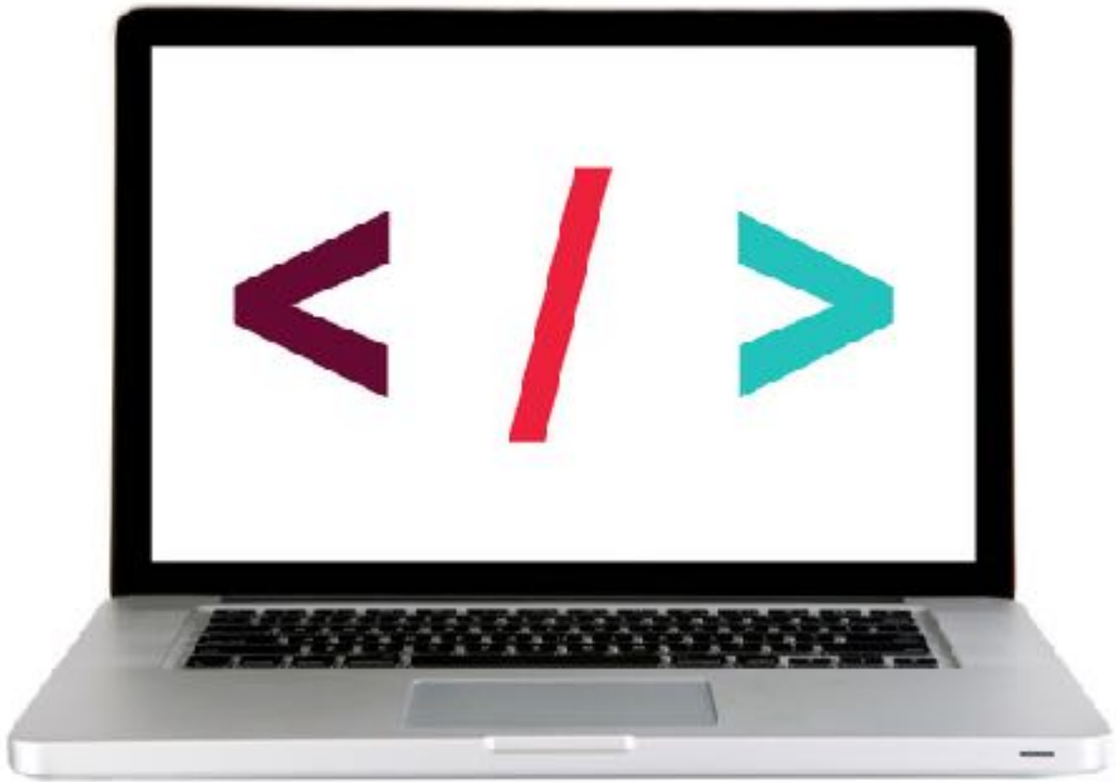
1. Copy the code from the codealong to the main.js file.
2. Change the URL to the one shown in the instructions.
3. Verify that a new set of results is shown in the console.
4. BONUS: Customize the error message to display the text of the HTTP status message.
   (Hint: look at https://developer.mozilla.org/en-US/docs/Web/API/Response/statusText)
5. BONUS: Refactor the code to work with user interaction. In the index.html file there is a "Get Health Data" button. Modify your code so data is only requested and logged to the console after a user clicks the button.

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

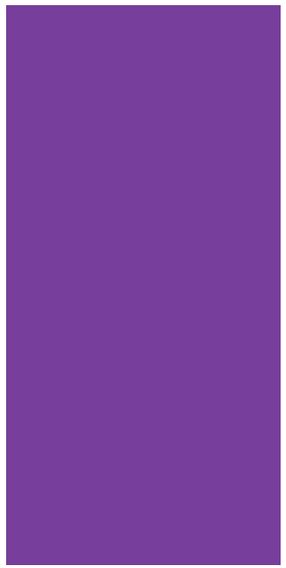
LET'S TAKE A CLOSER LOOK



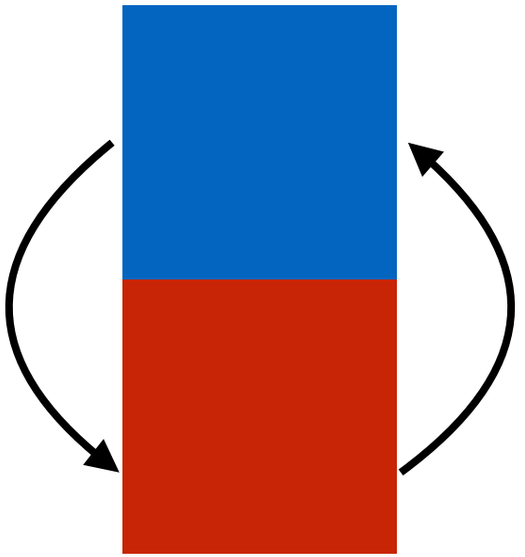
Code organization

SEPARATION OF CONCERNS

code for data and view intermingled



code for data

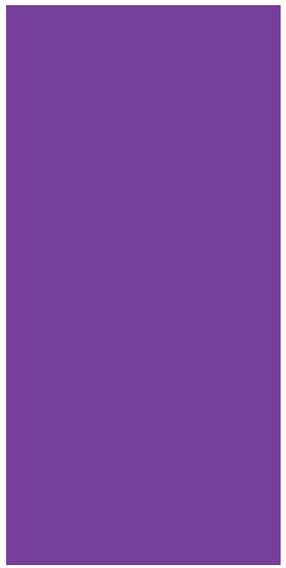


code for view

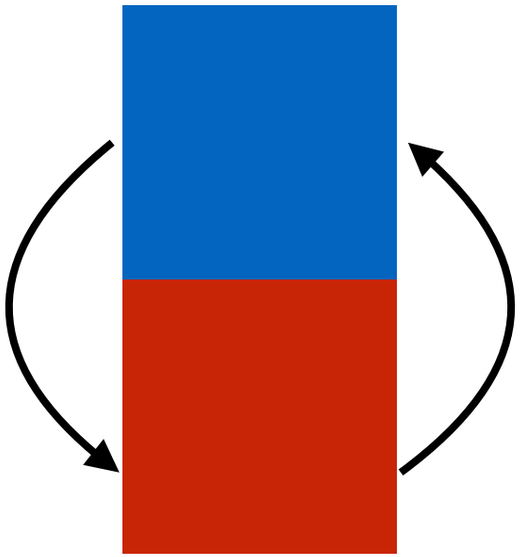
parts of code call each other, but are maintained separately

SEPARATION OF CONCERNS - HTTP

code for client and for HTTP requests intermingled



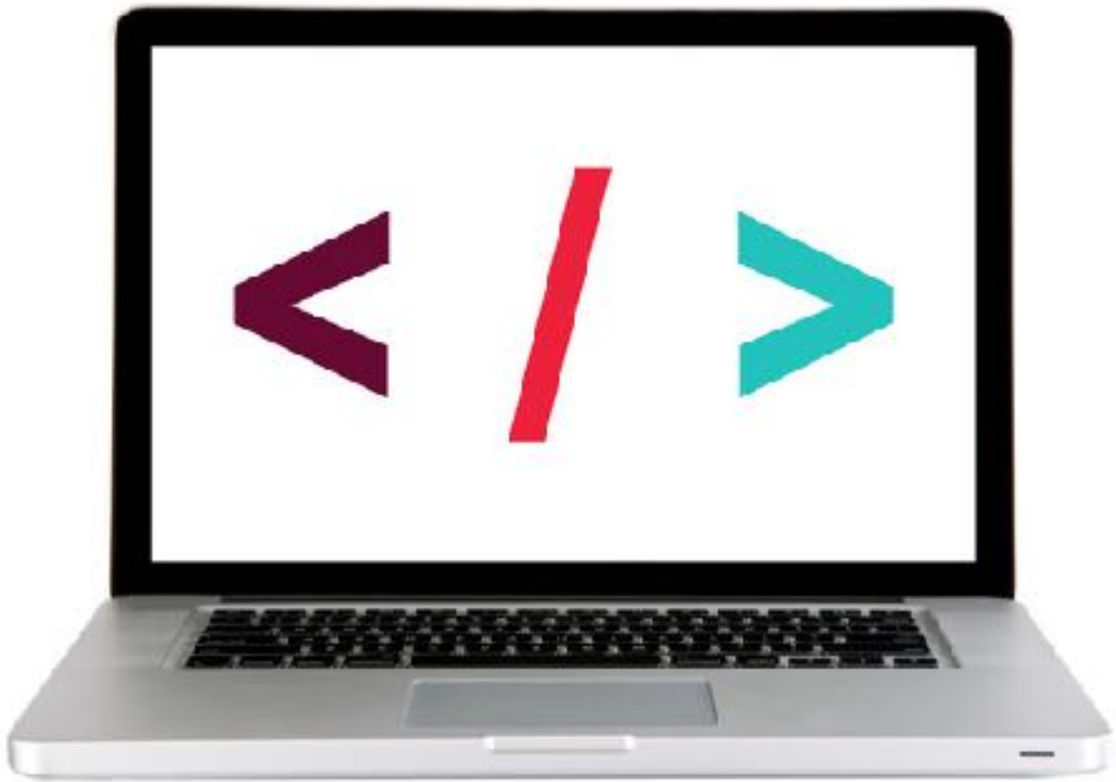
code for client



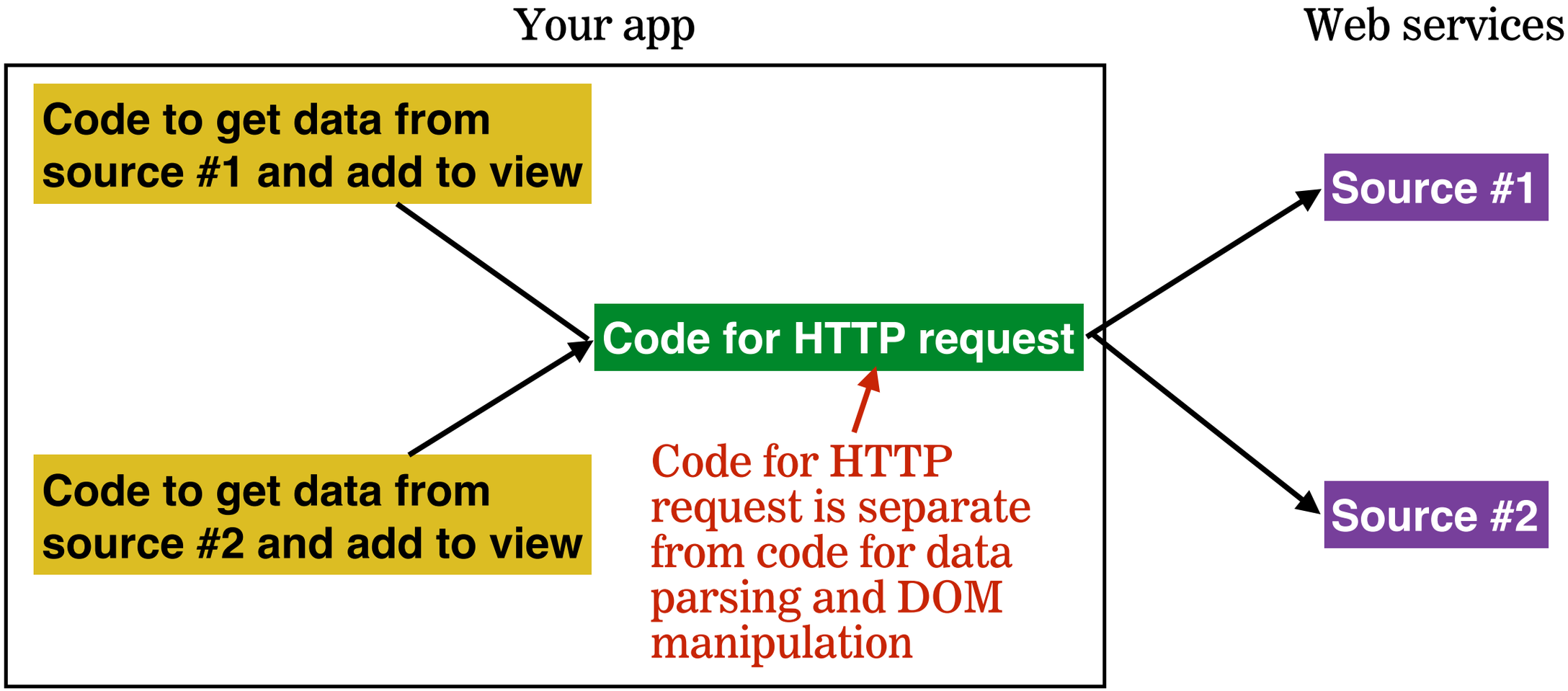
code for HTTP

parts of code call each other, but are maintained separately

LET'S TAKE A CLOSER LOOK



CREATING DRY CODE FOR HTTP REQUESTS



LAB — JQUERY AJAX

OBJECTIVE

- Create an Ajax request using jQuery or Fetch.

LOCATION

- `starter-code > 4-ajax-lab`

EXECUTION

45 min

1. Open index.html in your editor and familiarize yourself with the structure and contents of the file.
2. Open main.js in your editor and follow the instructions.

Asynchronous programming

WHAT WOULD YOU SEE IN THE CONSOLE?

```
let status;
function doSomething() {
    for (let i = 0; i < 1000000000; i++) {
      numberArray.push(i);
    }
    status = “done”;
    console.log(“First function done”);
}
function doAnotherThing() {
    console.log(“Second function done”);
}
function doSomethingElse() {
    console.log(“Third function: ” +
status);
}
```

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

WHAT WOULD YOU SEE IN THE CONSOLE?

```
let status;
function doSomething() {
    for (let i = 0; i < 1000000000; i++) {
      numberArray.push(i);
    }
    status = “done”;
    console.log(“First function done”);
}
function doAnotherThing() {
    console.log(“Second function done”);
}
function doSomethingElse() {
    console.log(“Third function: ” +
status);
}
```

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

```
// result in console
// (after a few seconds):
> “First function done”
> “Second function done”
> “Third function: done”
```

SYNCHRONOUS CODE

- Statements are executed in order, one after another
- Code blocks program flow to wait for results
- Most JS code is synchronous

ASYNCHRONOUS CODE

- Code execution is independent of the main program flow
- Statements are executed concurrently
- Program does not block program flow to wait for results
- Certain JS statements are asynchronous by default

https://en.wikipedia.org/wiki/Asynchrony_(computer_programming)

ASYNCHRONOUS PROGRAM FLOW

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

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

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

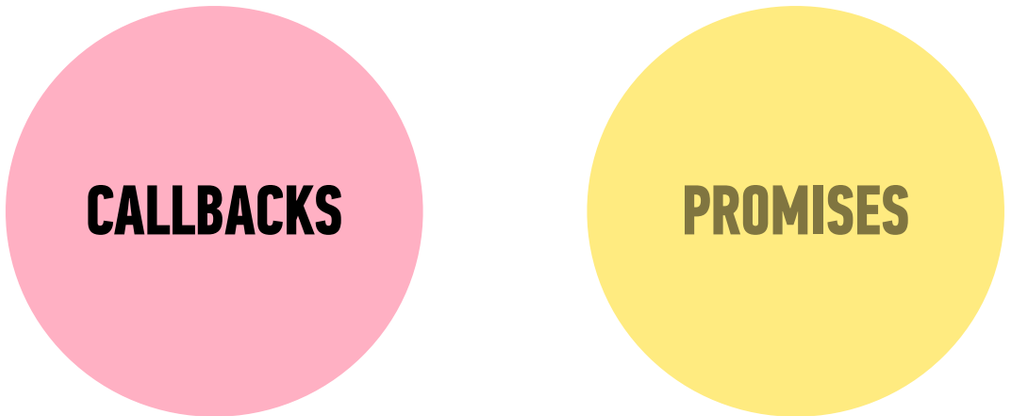
APPROACHES TO ASYNCHRONOUS PROGRAM FLOW

Functions & callbacks

HOW MANY ARGUMENTS IN THIS CODE?

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

APPROACHES TO ASYNCHRONOUS PROGRAM FLOW



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(() => {
  console.log("Hello world");
}, 1000);
```

SETTIMEOUT WITH NAMED FUNCTION ARGUMENT

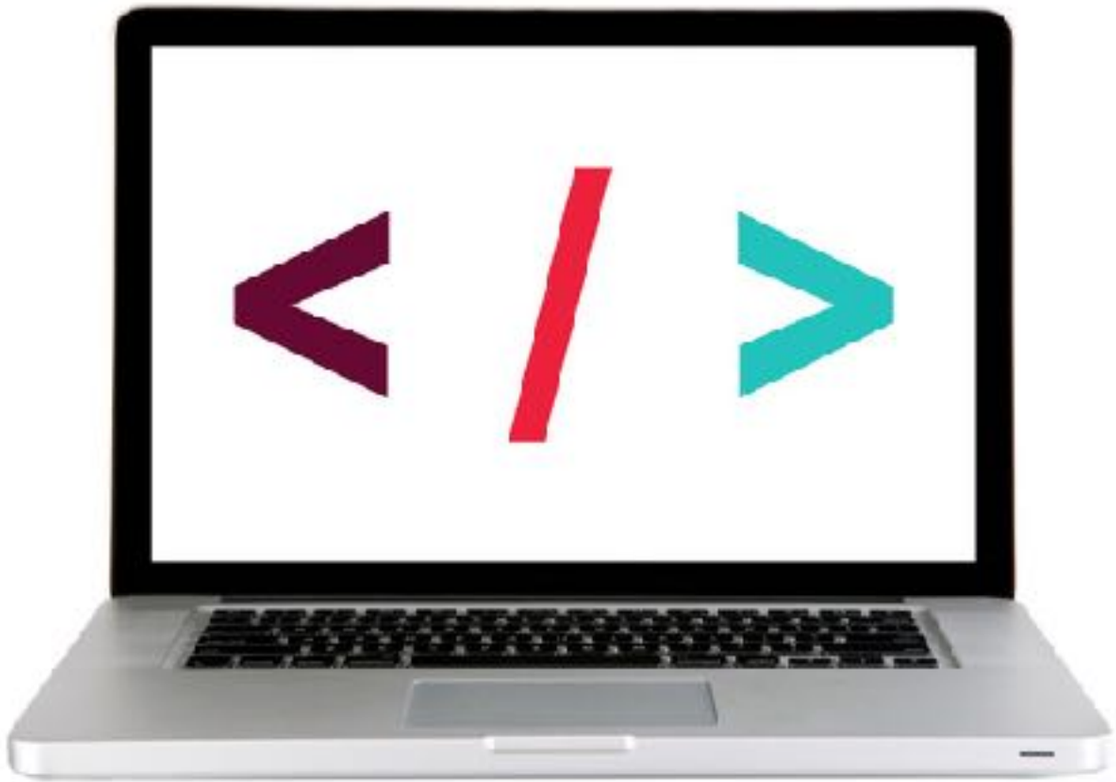
```
const helloWorld = () => {
  console.log("Hello world");
}

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

LOCATION

- `starter-code > 6-callback-exercise`

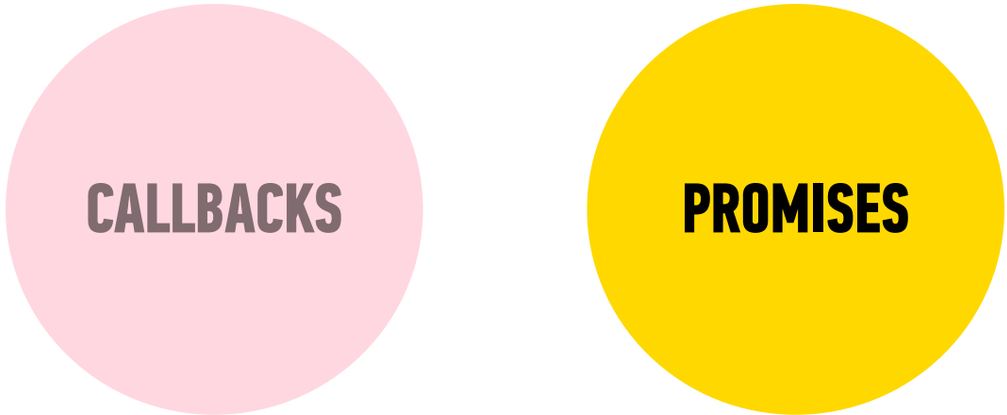
TIMING

20 min

1. In your editor, open script.js.
2. Follow the instructions to create the add, showAnswer, calcResult, and subtract functions, and to call the calcResult function using the add and subtract functions as callbacks.
3. Test your work in the browser and verify that you get the expected results.
4. BONUS: Update the showAnswer function to change the content of the element with the id value 'operator' to a plus symbol after the user clicks the Add button, or to a minus symbol after the user clicks the Subtract button.

Promises & Fetch

APPROACHES TO ASYNCHRONOUS PROGRAM FLOW



PROMISES

traditional callback:

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

callback using a promise:

```
doSomething()
  .then((result) => {
    // work with result
  })
  .catch((error) => {
    // handle error
  });
```

MULTIPLE CALLBACKS — TRADITIONAL CODE

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

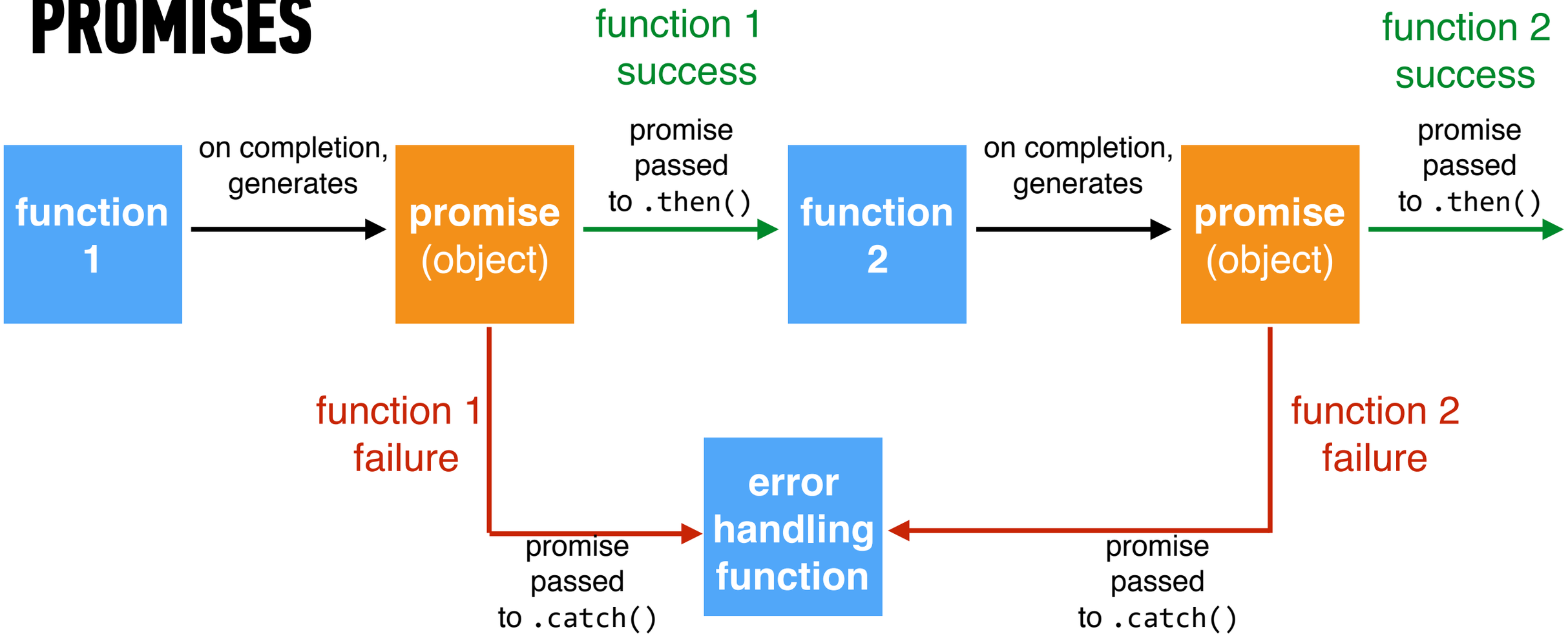
MULTIPLE CALLBACKS WITH PROMISES

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

ERROR HANDLING WITH PROMISES

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

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

Fetch

```
fetch(url)
  .then((res) => {
    if(res.ok) {
      return res.json();
    } else {
      throw ‘problem';
    }
  })
  .then((data) => {
    // DOM manipulation
  })
  .catch((error) => {
    // handle lack of data in UI
  });
```

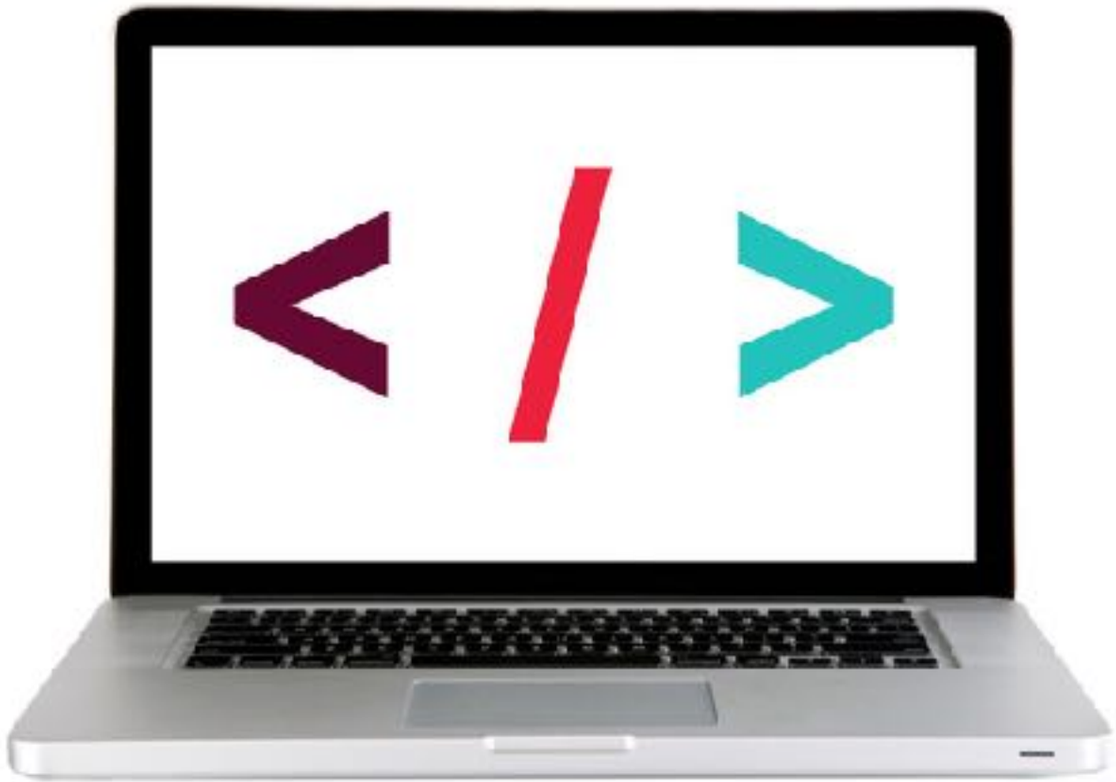
jQuery .get()

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

LET'S TAKE A CLOSER LOOK



Exit Tickets!

(Class #9)

LEARNING OBJECTIVES - REVIEW

- Access public APIs and get information back.
- Implement an Ajax request with Fetch.
- Create an Ajax request using jQuery.
- Describe what asynchronous means in relation to JavaScript
- Pass functions as arguments to functions that expect them.
- Write functions that take other functions as arguments.
- Build asynchronous program flow using Fetch

NEXT CLASS PREVIEW

Asynchronous JavaScript and Callbacks

- Pass functions as arguments to functions that expect them.
- Write functions that take other functions as arguments.
- Build asynchronous program flow using Fetch
- Integrate string and variable values using template literals

Q&A