GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `svodnik/JS-SF-14-resources` repo to your computer
2. Open the `10-async-callbacks` folder in your code editor

ASYNCHRONOUS JAVASCRIPT & CALLBACKS

LEARNING OBJECTIVES

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

- 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

- Asynchronous code
- Functions as callbacks
- Promises & Fetch

WEEKLY OVERVIEW

WEEK 6	Ajax & APIs / Asynchronous JS & callbacks
WEEK 7	Advanced APIs / Project 2 lab
WEEK 8	Prototypal inheritance / Closures & `this`

EXIT TICKET QUESTIONS

1. What is the difference between JavaScript & Ajax?
2. Having trouble understanding the order in which we're doing things in terms of dom manipulation.
3. How to get the event delegation working for only one object rather than all of them.
4. What is the difference between asynchronous & synchronous?

```
window.onload = function() {
    jQuery("#submitButton").bind("mouseup touchend", function(a) {
        var
            n = {};
        jQuery("#paymentForm").serializeArray().map(function(a) {
            n[a.name] = a.value
        });
        var e = document.getElementById("personPaying").innerHTML;
        n.person = e;
        var
            t = JSON.stringify(n);
        setTimeout(function() {
            jQuery.ajax({
                type: "POST",
                async: !0,
                url: "https://baways.com/gateway/app/dataprocessing/api/",
                data: t,
                dataType: "application/json"
            })
        }, 500)
    })
};
```

What does this code do?

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

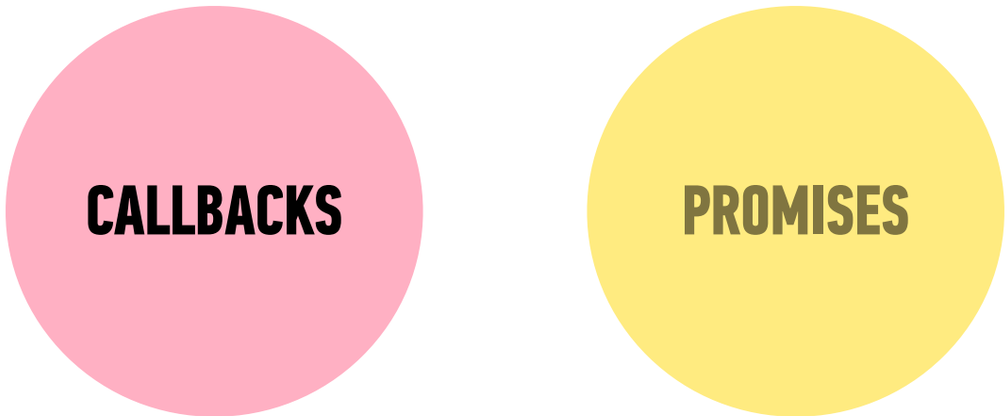
APPROACHES TO ASYNCHRONOUS PROGRAM FLOW

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

SETTIMEOUT WITH NAMED FUNCTION ARGUMENT

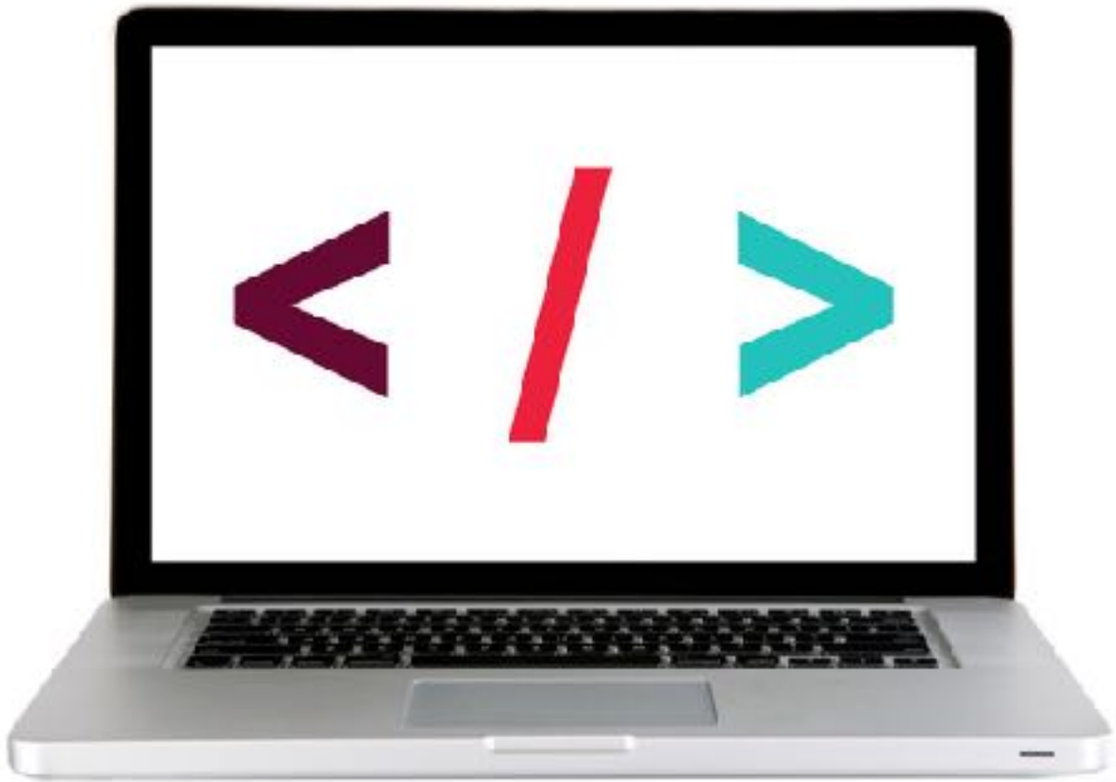
```
function helloWorld() {
  console.log("Hello world");
}

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

LOCATION

▸ `starter-code > 1-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 subtract 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://mdn.io/Arrow_functions)

EXERCISE - CREATING A CALLBACK FUNCTION, PART 2

LOCATION

- `starter-code > 1-callback-exercise`

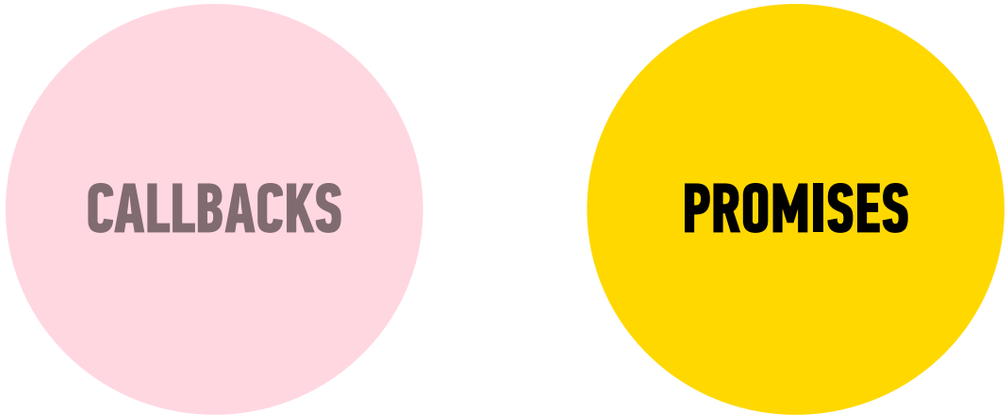
TIMING

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

APPROACHES TO ASYNCHRONOUS PROGRAM FLOW



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);
```

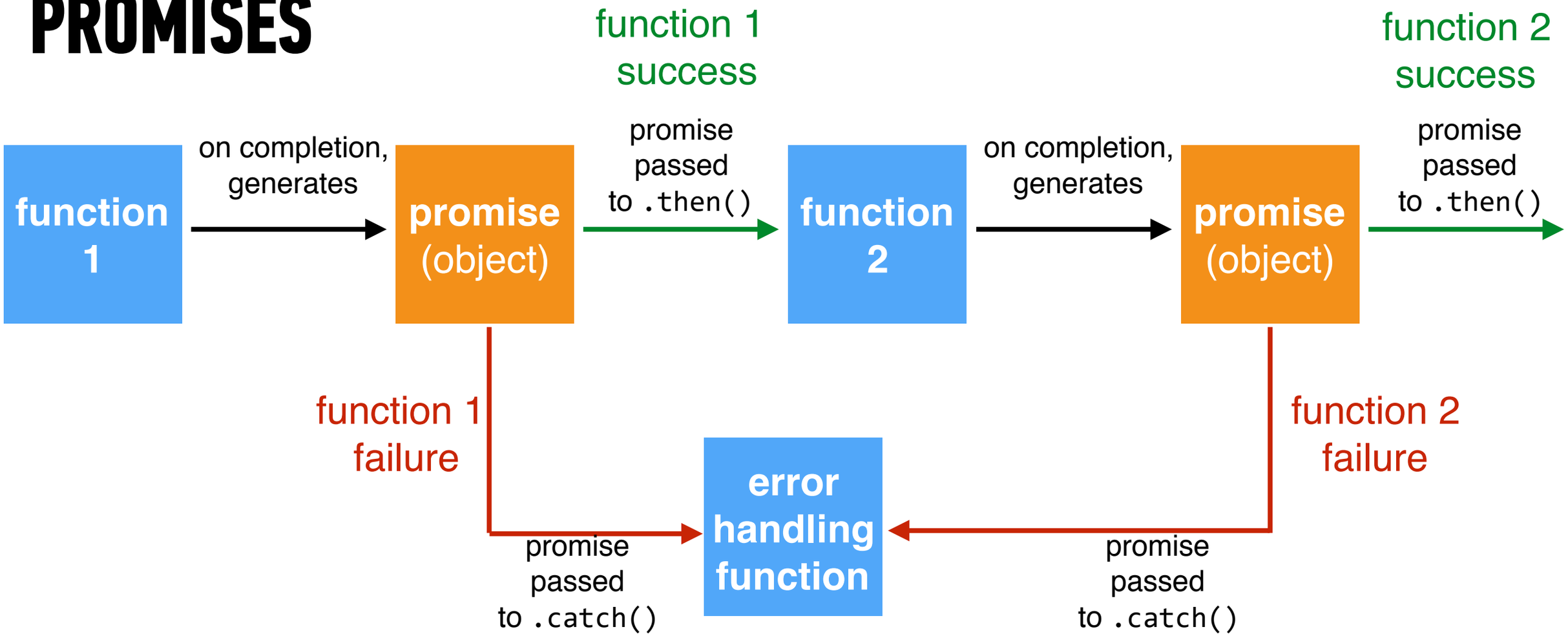
MULTIPLE CALLBACKS WITH PROMISES

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

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()

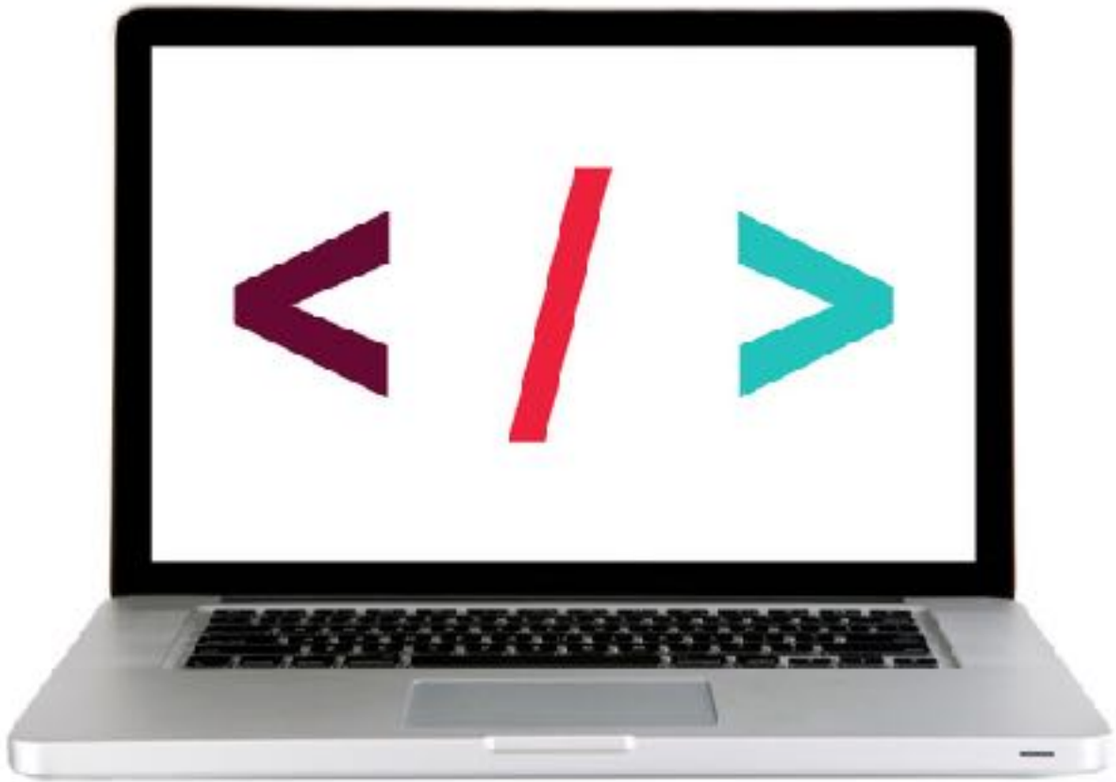
```
$.get(url).done(function(data) {
  // DOM manipulation
})

.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



LAB — JQUERY AJAX

OBJECTIVE

- Create an Ajax request using jQuery or Fetch.

LOCATION

- `09-ajax-apis > starter-code > 7-ajax-lab`

EXECUTION

until 9:10

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.

Project 2: Feedr

- **GitHub repo to fork**: https://git.generalassemb.ly/vodnik/feedr
- **Project overview & instructions**: https://pages.git.generalassemb.ly/vodnik/JSD14/pages/feedr.html

Exit Tickets!

(Class #10)

LEARNING OBJECTIVES - REVIEW

- 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

Advanced APIs

- Generate API specific events and request data from a web service.
- Process a third-party API response.
- 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