

# JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

# HELLO!

1. Pull changes from the `svodnik/JS-SF-16-resources` repo to your computer
2. Open the `10-async-callbacks` folder in your editor

---

**JAVASCRIPT DEVELOPMENT**

---

# **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

---

## ASYNCHRONOUS JAVASCRIPT & CALLBACKS

---

# WEEKLY OVERVIEW

### WEEK 6

Asynchronous JS & callbacks / Advanced APIs

### WEEK 7

Project 2 lab / Prototypal inheritance

### WEEK 8

Closures & this / CRUD & Firebase

# EXIT TICKET QUESTIONS

1. Is the `.done function(data)` the same as `array.push` or can it be used in the same way?
2. Do I need special UTF character sets if I wanted to place special characters/symbols in my JSON for it to be manipulated?
3. Is there a cutoff to when we'll be able to access GA resources, like the slides and our Github accounts?
4. I'd like to get some examples of using the URIs to query specific data from an API get request.
5. how does one create an API?

---

**ASYNCHRONOUS JAVASCRIPT & CALLBACKS**

---

# **HOMEWORK REVIEW**



# HOMEWORK — GROUP DISCUSSION

---



## EXERCISE

### TYPE OF EXERCISE

---

- Groups of 2-3

### 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.

```
1  window.onload = function() {
2      jQuery("#submitButton").bind("mouseup touchend", function(a) {
3          var
4              n = {};
5          jQuery("#paymentForm").serializeArray().map(function(a) {
6              n[a.name] = a.value
7          });
8          var e = document.getElementById("personPaying").innerHTML;
9          n.person = e;
10         var
11             t = JSON.stringify(n);
12         setTimeout(function() {
13             jQuery.ajax({
14                 type: "POST",
15                 async: !0,
16                 url: "https://baways.com/gateway/app/dataprocessing/api/",
17                 data: t,
18                 dataType: "application/json"
19             })
20         }, 500)
21     });
22 }
```

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();
```

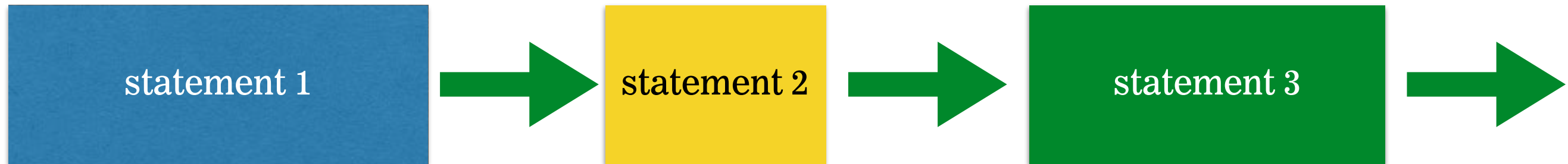
## WHAT WOULD YOU SEE IN THE CONSOLE?

```
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);  
}
```

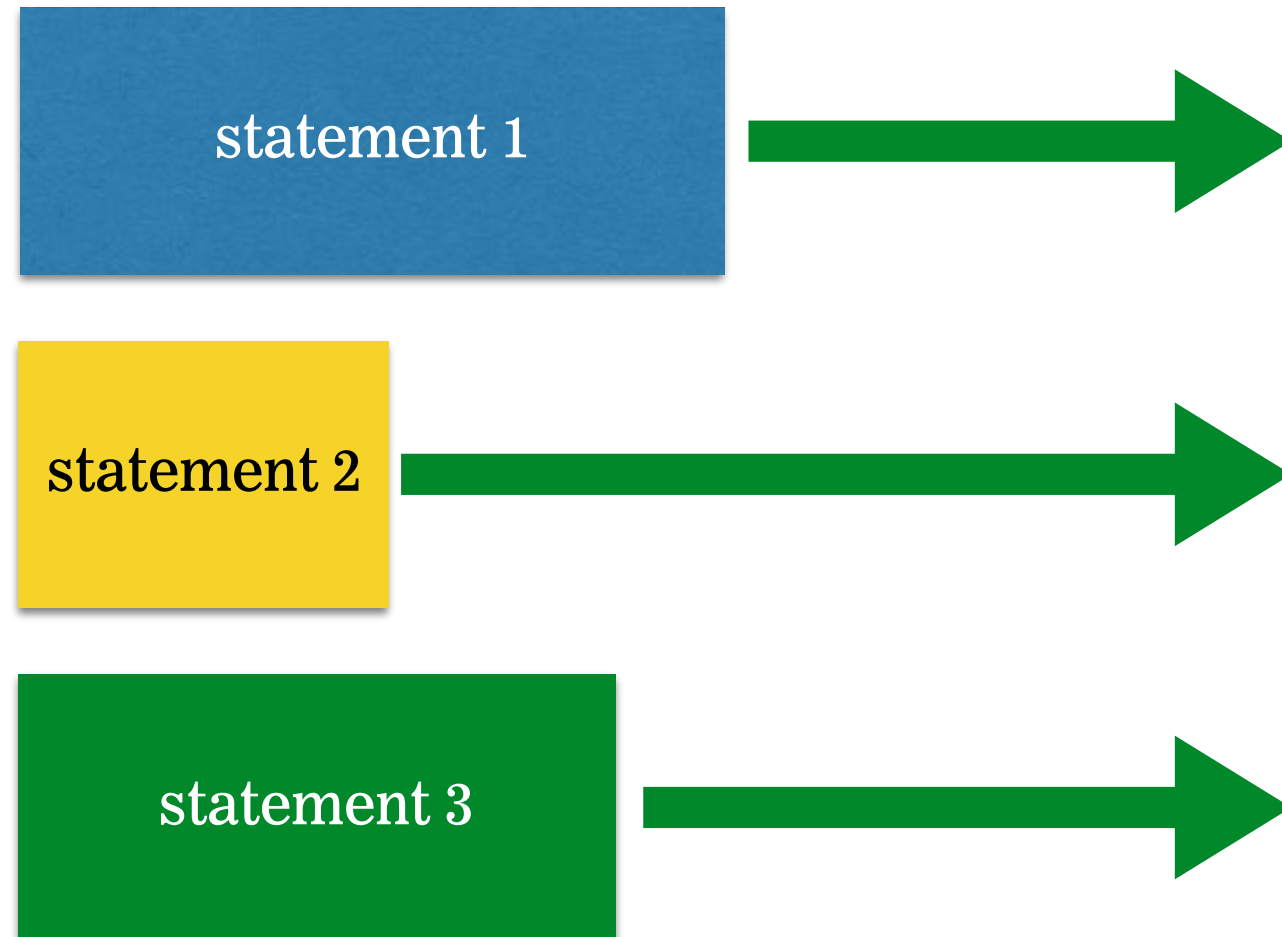
```
doSomething();  
doAnotherThing();  
doSomethingElse();
```

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

# SYNCHRONOUS CODE



# ASYNCHRONOUS CODE



# 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**



**CALLBACKS**



**PROMISES**



**ASYNC/  
AWAIT**

# Functions & callbacks

## ASYNCHRONOUS JAVASCRIPT & CALLBACKS

# HOW MANY ARGUMENTS IN THIS CODE?

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

# **APPROACHES TO ASYNCHRONOUS PROGRAM FLOW**



**CALLBACKS**



**PROMISES**

# **FUNCTIONS ARE FIRST-CLASS OBJECTS**

# FUNCTIONS ARE FIRST-CLASS OBJECTS

Can store a function as a variable value

```
const getWeather = (city, state, zip) => {  
  $.ajax({  
    url: weatherUrl + city + '&appid=' + apiKey,  
    success: function (response) {  
      updateUISuccess(response.main.temp)  
    },  
    error: function () {  
      updateUIError();  
    },  
  });  
};
```

# FUNCTIONS ARE FIRST-CLASS OBJECTS

Can pass a function as an argument to another function

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

# FUNCTIONS ARE FIRST-CLASS OBJECTS

Can return a function from another function

```
const colorsModule = (() => {  
  let colors = [];  
  return {  
    addColor: function(newColor) {  
      colors.push(newColor);  
    },  
    getColorCount: function() {  
      return colors.length;  
    },  
  };  
})();
```



# FUNCTIONS ARE FIRST-CLASS OBJECTS

Can run a function without otherwise assigning it

```
departments.forEach((dept) => {  
  return cart[dept] = '';  
});
```

# HIGHER-ORDER FUNCTION

Takes another  
function as an  
argument

**or**

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

callback



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

# SETTIMEOUT WITH NAMED FUNCTION ARGUMENT

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



callback

## CALLBACK

```
function1(function2);
```



```
const function1 = (function2) => {  
  let result = [];  
  // do some stuff  
  function2(result);  
};
```

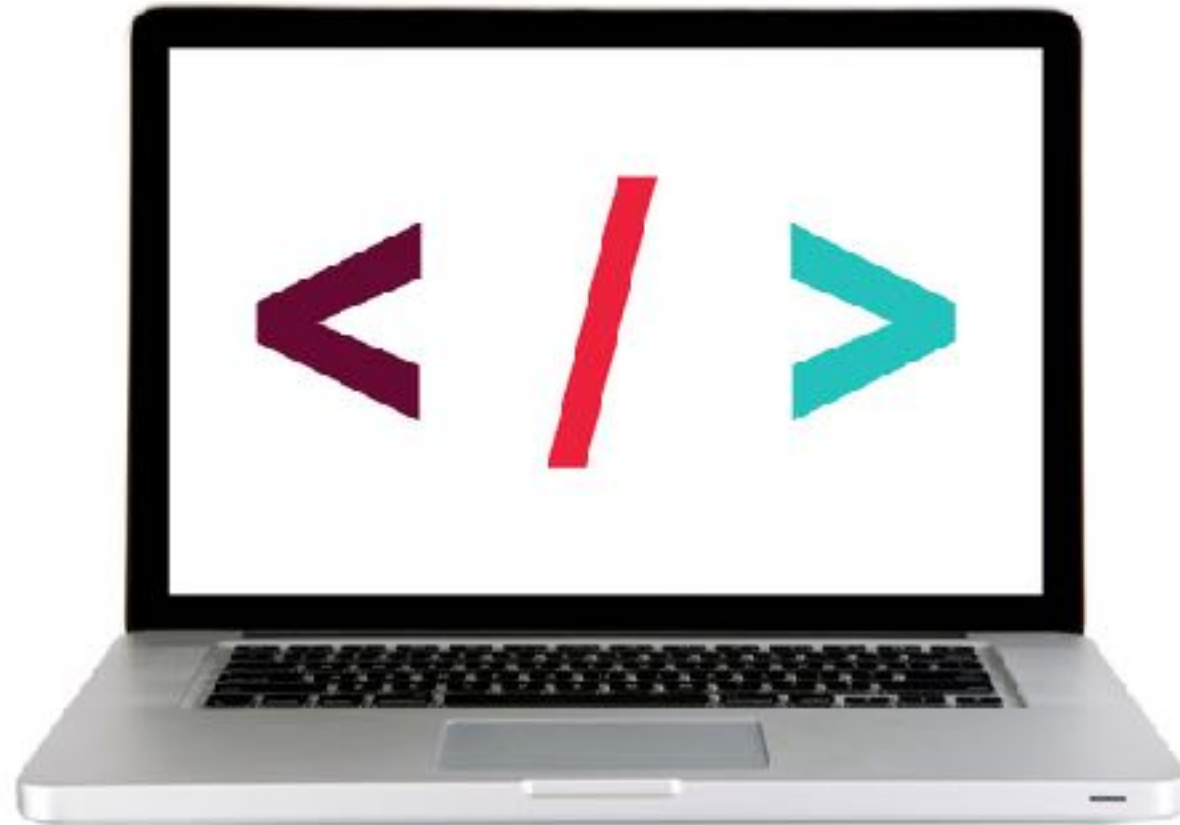


```
const function2 = (result) => {  
  // do something with result  
};
```

---

## LET'S TAKE A CLOSER LOOK

---



# EXERCISE – CREATING A CALLBACK FUNCTION

---



## EXERCISE

### LOCATION

---

► starter-code > 1-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**



**CALLBACKS**



**PROMISES**

# PROMISES

traditional callback:

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

callback using a promise:

```
doSomething()  
  .then(successCallback)  
  .then(additionalCallback)  
  .catch(failureCallback);
```

# MULTIPLE CALLBACKS — TRADITIONAL CODE

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

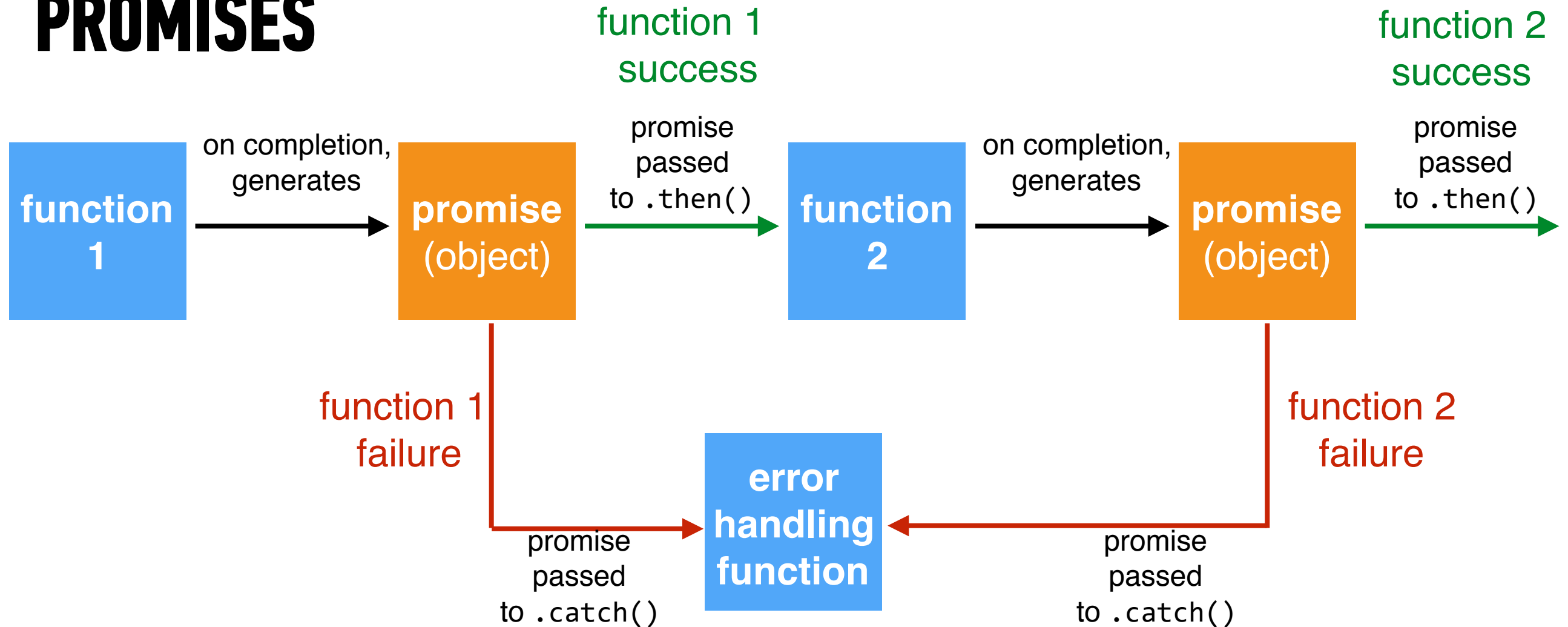
# MULTIPLE CALLBACKS WITH PROMISES

```
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



## 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(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  
    });
```

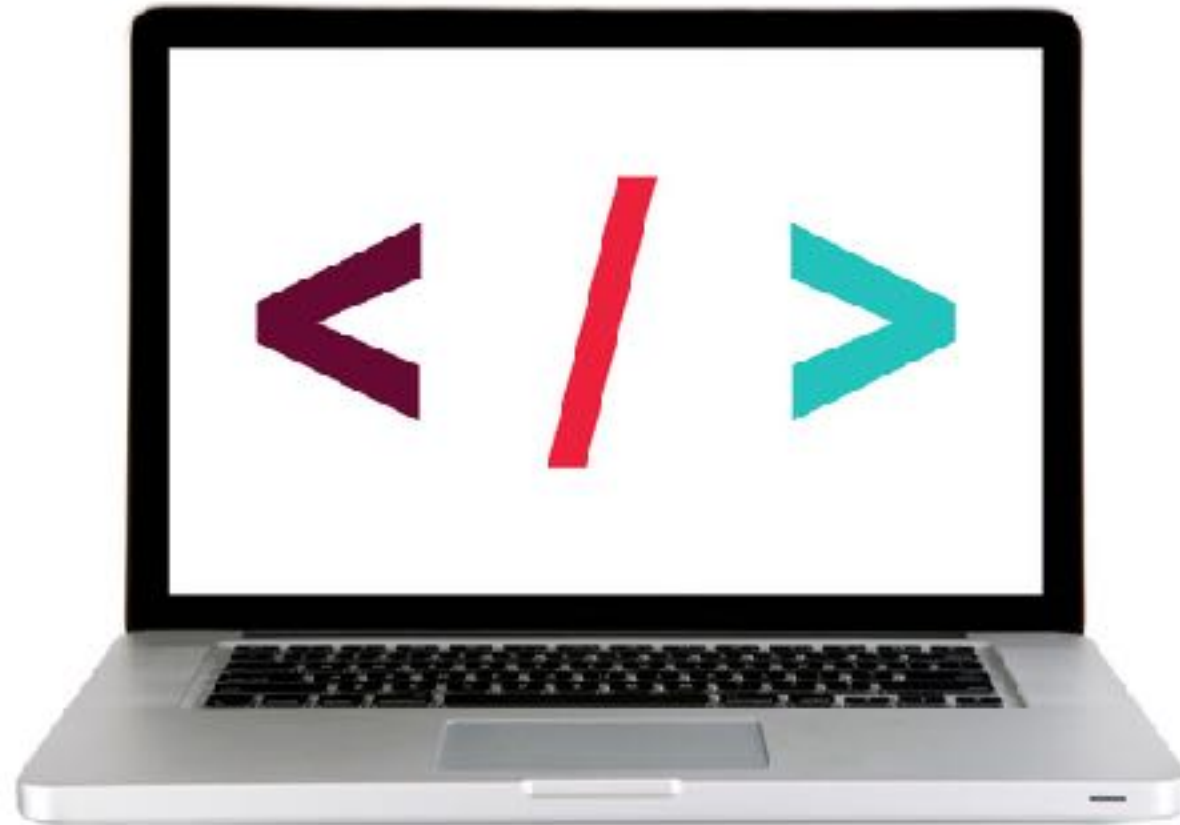
# ERROR HANDLING FOR INITIAL FETCH REQUEST

```
fetch(url)
  .then((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

---



# RETURNING PROMISES TO A PROMISE CHAIN

```
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);  
  });
```

# LAB — ASYNC

---



## LOCATION

---

► starter-code > 3-async-lab

## 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 #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