# About the workshop deck

Slides 1-3 of this deck are for **instructors only**.
Begin displaying slides to students at Slide 4.

# Versions

Numbering scheme
Major re-drafts (document-wide, or affecting majority of slides): Increment first digit, e.g. 1.0, 2.0
Minor updates that affect slides visible to students: Increment second digit, e.g. 1.1, 1.2
Minor updates that affect slides visible to instructors/notes only: Increment third digit, e.g. 1.1.1, 1.1.2

| Version | Date | Author | Slides & Changes |
|---|---|---|---|
| 1.0 | 2019 | Minae Lee | First version |
| 1.1 | 2020 | Steve Thompson | |
| 1.2 | March 3, 2022 | Minae Lee | Added more JavaScript slides to reflect curriculum changes (looping through arrays, functions, events), updated overall style |
| | | | |
| | | | |
| | | | |
| | | | |

# Instructor setup

- Have a list of your students ready so that you can check it against those who attend the workshop.
- Log in to your Zoom meeting.
- Log into Slack. Some students may need to reach you this way!
- Share the Zoom meeting link in Slack.
- Log in to the Zoom meeting earlier than your students.
- As students arrive, greet them and ask them to log in to Slack and the learning portal.
- Make sure your camera is on, and encourage students to turn on theirs.


# Bootstrap:

## Week 4 Workshop Presentation

# Workshop Agenda

| Activity | Estimated Duration |
| --- | --- |
| Set Up & Check-In | 15 mins |
| Week 4 Review | 60 mins |
| Assignment | 45 mins |
| Break | 15 mins |
| Assignment | 90 mins |
| Check-Out (Feedback & Wrap-Up) | 15 mins |

# Set up

Along with Zoom, please also be logged into Slack and the learning portal!

# Week 4 Review

# Overview

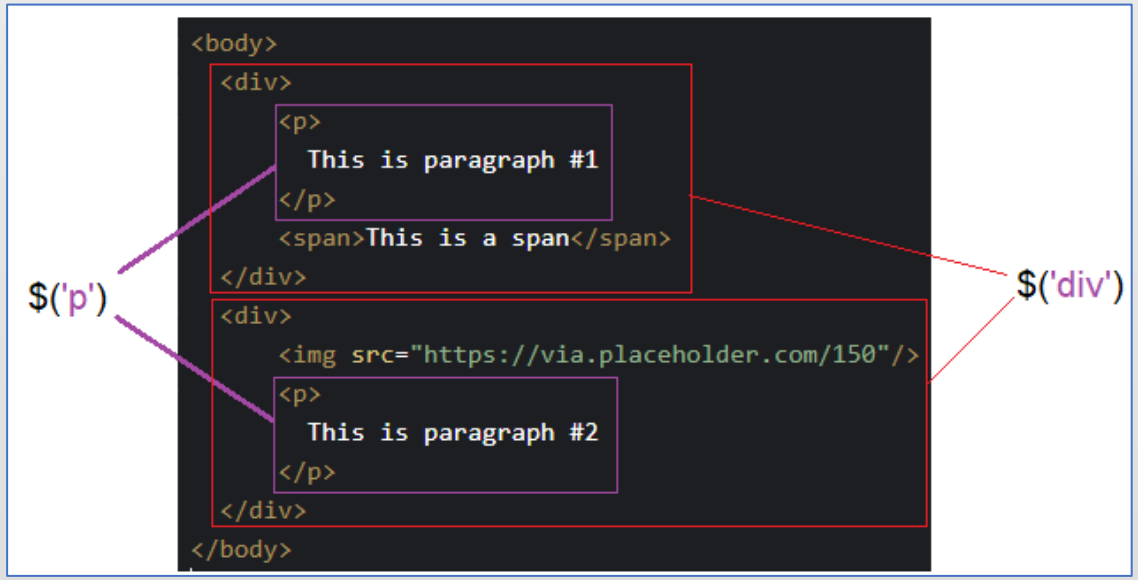
| | |
|---|---|
| jQuery | JavaScript Arrays |
| Scss | JavaScript Array Methods |
| NPM Scripts | Looping through Arrays |
| Build Processes | Function Declarations |
| JavaScript While Loops | Events |
| JavaScript For Loops | |

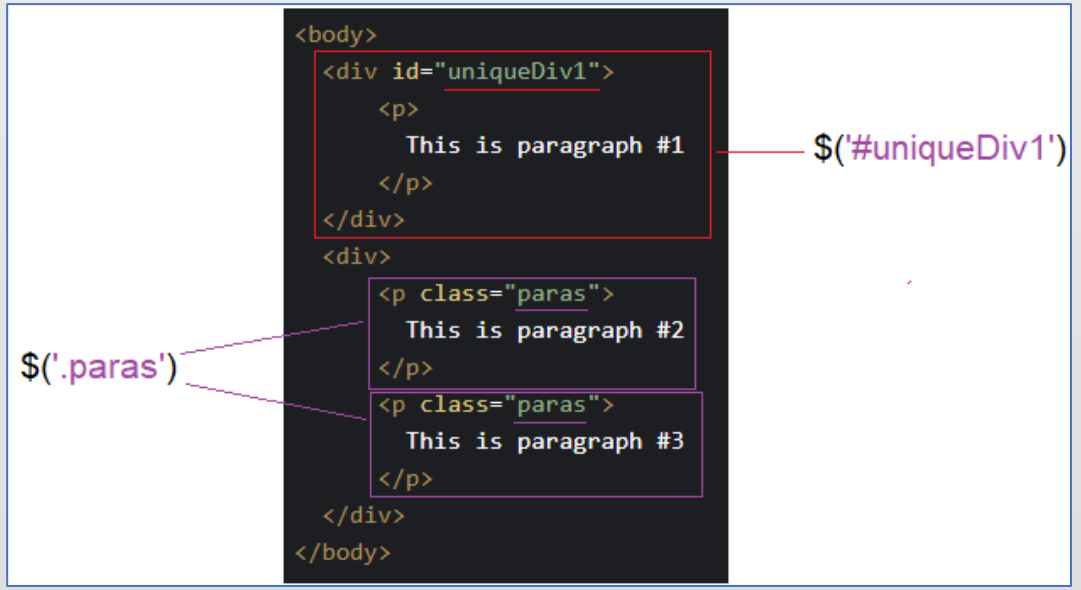
# Review: jQuery

## Discuss:

- How do you use jQuery to select a specific DOM node?

```
$('css selector')
```



# Review: jQuery

## Discuss:

- What is the way to activate a jQuery script in an HTML document when the document loads?

**NOTE:** This code that we are using in the beginning of the jQuery script:

```
$(function() { ... });
```

is the shorthand, recommended version for this code, called the jQuery *ready method* or *ready function*, which specifies a function to load when the webpage's DOM is ready:

```
$( document).ready(function() { ... });
```

# Review: Scss

## Discuss:

- What is the difference between Sass and Scss?

SCSS is a variant of Sass which is a CSS preprocessor (compiles Sass code to CSS syntax)

- You used variables and nesting in your Sass exercise. What are some of the other ways you can use Sass?

Creating Variables
$color-dark: #60106b;

Utilize built-in functions
lighten(color, amount)

Utilize custom functions
@mixin

Looping through elements
@for @each

Logically separate style code into multiple files for manageability
@_partials

# Review: NPM Scripts (1 of 3)

## Discuss:

- Go through each of the scripts inside the scripts object in package.json and name what each one does.

```
"build": "npm run clean && npm run imagemin && npm run copyfonts && npm run usemin"
```

**build:** Runs "clean" then "imagemin" then "copyfonts" and finally the "usemin" npm scripts

```
"clean": "rimraf dist"
```

**clean:** Runs node package "rimraf" on the "**dist**" folder which will delete it

```
"copyfonts": "copyfiles -f node_modules/font-awesome/fonts/* dist/fonts"
```

**copyfonts:** Runs node package "copyfiles" and copies font-awesome fonts to "**dist/fonts**" folder

```
"imagemin": "imagemin img/* -o dist/img"
```

**imagemin:** Runs node package "imagemin" to compress all files in "**img**" and outputs compressed files to "**dist/img**" folder

# Review: NPM Scripts (2 of 3)

```
"usemin": "usemin contactus.html -d dist --htmlmin -o dist/contactus.html && usemin aboutus.html
          -d dist --htmlmin -o dist/aboutus.html && usemin index.html -d dist --htmlmin -o dist/index.html",
```

**usemin:** Runs node package "usemin" to compress html & css files and output to "**dist**" folder

```
"lite": "lite-server",
```

**lite:** Runs node package "lite-server" which runs local web server for development purposes

```
"scss": "node-sass -o css/ css/",
```

**scss:** Runs node package "node-sass" which compiles all .scss files in "css" folder and outputs compressed version to "dist/css" folder

```
"start": "npm run watch:all",
```

**start:** Runs script "watch:all"

```
"watch:scss": "onchange \"css/*scss\" -- npm run scss",
```

**watch:scss:** Runs node package "onchange" which watches all *.scss (SASS) files located in "**css**" folder. If a change is made, it will run "scss" script

```
"watch:all": "parallelshell \"npm run watch:scss\" \"npm run lite\""
```

**watch:all:** Runs node package "parallelshell" which allows multiple terminals to be open concurrently then runs "watch:scss" & "lite" scripts

# Review: NPM Scripts (3 of 3)

What does the && symbol do in these scripts?

You learned that the **&&** operator returns with the first **falsy** value it runs into, or the last **truthy** value if there are no falsy values.

The principle is the same when applied to the scripts

```
"build": "npm run clean && npm run imagemin && npm run copyfonts && npm run usemin"
```

The "**build**" script will first try "npm run clean", then if it works it will go to the next script, "npm run imagemin", and so forth, but if "npm run clean" fails (is falsy) it will abort.

# Review: Build Processes

**Discuss:**

- What is minification?

  **The process of removing unncessary code in order to reduce the file size and increase load times**

- What is uglification?

  **Compressing the files size by removing all white spaces, new lines, and other performance enhancements at the cost of making the code "ugly" and reduced readability**

- What is file concatenation? (in the context of building your files for deployment)

  **The process of combining multiple files into one single file for reference**

# Review: JavaScript While Loops

## Discuss:

- What is the difference between a **do ... while** loop and a **while** loop?

```
let i = 0;

while (i < 10) {
    console.log("The iterator is", i);
    i++;
}
```

Runs AT LEAST 1 time

```
i = 0;

do {
    console.log(i);
    i++;
} while (i > 0 && i < 10);
```

# Review: JavaScript For Loops

- What is wrong with this for loop, and how would you fix it so that it prints the numbers 5-10 to the console?

**WARNING: DO NOT** type the code below into your console as-is. Discuss it first with your instructor and fix the code before testing it in your console.

```
for (let i = 11; i > 10; i++) {
    console.log(i);
}
```

**This code will cause an "infinite loop" and eventually crash when memory is filled**

```
for (let i = 5; i <= 10; i++) {
    console.log(i);
}
```

# Review: JavaScript Arrays

**Discuss:**

- What is an array, and how do you create one?

A single variable that stores multiple items
const fruits = ['mangos', 'bananas', 'pineapples']

You can declare an array with '**let**' as well

- How would you access the second item in an array?

**fruits[1]**
Remember that arrays use a zero-based index (starts at 0)

- How do you find the length of an array?

**fruits.length**

# Review: JavaScript Array Methods

## Discuss:

- What does the word *method* mean in JavaScript?

  **A method is a function that is tied to an object (an array is a special type of object)**

- What does the syntax to call a method typically look like?

```
object.method(argument);
```

- There is more than one way to check if an item exists inside an array. Name two array methods that can do this. What are the differences?

**indexOf()**

**Takes an argument and returns its index # if exists or –1 if not in the array**

```
myArr.indexOf('foo');
```

**includes()**

**Takes an argument and returns a boolean true if exists or false if not in the array**

```
myArr.includes('bar');
```

# Review: JavaScript Array Methods

## Discuss:

- What is a **destructive/mutating** method vs. a **non-mutating** method?

**A destructive/mutating method will modify the original array ( e.g. push() )**

**A non-mutating method will not modify the original array and create a new array in memory ( e.g. concat() )**

- What are the differences between push/pop and shift/unshift?

**push/pop (add/remove) will mutate the end of the original array**

**shift/unshift (add/remove) will mutate the beginning of the original array AND shift the index value of the items in the array down**

- What are the return values of push, pop, shift, and unshift?

**push - returns the new length of the array (array.length)**
**pop - returns the value that was removed from the end of the array**

**shift - returns the value that was removed from the beginning of the array**
**unshift - returns the new length of the array (array.length)**

# Review: Looping through Arrays

```
const sports = ['baseball', 'football', 'water polo'];
```

Using a standard **for** loop to iterate through the **sports** array:

```
for (let i = 0; i < sports.length; i++) {
    // do something with each item in the array
    console.log(sports[i])
}
```

Using a **for ... of** loop to iterate through the **sports** array:

```
for (const sport of sports) {
    // do something with each item in the array
    console.log(sport)
}
```

# Review: Using Arrays

- Students: Open your browser's Developer Console and type along:

  - `let myArr = [1,2,3,4];`
  - `console.log(myArr);`
  - `myArr.push(5,6,7);` [Returns the new length of the array]
  - `console.log(myArr);`
  - `console.log(myArr[2]);` [Logs the item at index 2, which is the number 3]
  - `myArr[2] = 33;` [This updates the value at index 2 to 33]
  - `console.log(myArr);` [Notice that the 3 has been updated to 33]
  - `myArr.pop();` [Returns 7 which was in the last index of the array]
  - `console.log(myArr);` [Notice that the 7 is now gone from the array]
  - `myArr.shift();` [Returns the 1 which was at the first index of the array]
  - `console.log(myArr);` [Notice that the 1 is now gone from the array]
  - `myArr.unshift(0,1);` [Returns the new length of the array]
  - `console.log(myArr);` [Notice that 0 and 1 are added to the beginning of the array]

- If you don't understand what each line does, or why it does what it does, please ask for clarification!

# Review: Function Declarations (1 of 2)

- Functions are reusable units of code that can take values as inputs (**arguments**), and return a value as an output (**return value**)
- There are multiple ways to define functions in JavaScript
- Function declaration example:

```
function hasFoo(word) {
    if (word.includes("foo")) {
        return true;
    }
    return false;
}
```

Function call/invocation example:

```
> hasFoo("abcdfoo")
true
> hasFoo("abcdefg")
false
```

- Discuss:
  - In the function declared above, what is the parameter list?
  - What is the return value?
  - In the two example function calls (a.k.a. function invocations) shown to the right of the function declaration, what part is the argument list?
  - Do you understand how each of the two function calls resulted in its return value?
  - If a function does not have any inputs (no parameters), does it still require a parameter list in the function declaration? Does it still require an argument list when the function is called?

# Function Declarations (2 of 2)

## Discuss:

- What does it mean to say that function declarations are "hoisted"?

  Hoisted means that at runtime, JS will move all function declarations to the top of their scope

- In the code below, which is the parameter and which is the argument?

```
function inchesToCM(inches) {
    return inches * 2.54;
}

const lengthInches = 3;
const lengthCm = inchesToCM(lengthInches);
```

Argument = lengthInches
(pass variables/values to functions)

Parameter = inches
(take in arguments as local variable)

JS would throw an error that "inches" is not defined since it's only defined in the function's scope

# Review: Events (1 of 2)

- We can write code for a webpage to automatically respond to "events" such as a user clicking on an element, pressing a key, etc.
- Two main ways: Event handlers (a.k.a. on*event* handlers) and event listeners
- Event handlers can be written inline in HTML page:

```
<span onmouseover="console.log('mouse moved over me')">DEMO</span>
```

- Event handlers can also be written in JavaScript file attached to HTML page:

```
document.querySelector('span').onmouseover = console.log('mouse moved over me');
```

- Event listeners (preferred, more modern) are also written inside JavaScript:

```
document.querySelector('span').addEventListener('mouseover', 'mouse moved over me');
```

# Review: Events (2 of 2)


Discuss:

- In the following inline onevent handler: `onclick="alert('Hello')"`
  - Why is it necessary to use single quotes around the 'Hello!' Instead of double?

To prevent JavaScript from mistaking "alert(" as a string, since once a single or double quote is opened, it will look for another corresponding single or double quote to close the string

- What is wrong with the following code? Assume that there is a valid button element and a valid function named runSomeCode.

```
document.querySelector("button").onclick="runSomeCode()";
```

The onclick JS method is used incorrectly.
Should be .onclick=runSomeCode; (no quotes or parenthesis)

- What is the main difference between using the on*event* handlers (such as the onclick event handlers shown above) vs event listeners?

Even listeners are the more modern way to implement event handlers and support more than 1 event listener to an element

```
// Multiple listeners can be added to the same event and element
button.addEventListener('click', changeText)
button.addEventListener('click', alertText)
```

# Week 4 Workshop Assignment

- All students should aim to finish and submit your assignment before you leave today.
- Work in pairs, or groups of three. Talk to each other and figure things out together!
- 10-minute rule during workshops: If you and your paired partner have spent more than 10 minutes trying to figure something out, ask your instructor for help.

# Happy learning!

# Review: Challenges/Quiz

- It is important that students have ample time to complete the assignment during the workshop.
- If there is time left *after* students have completed the Workshop Assignment, review the Week 4 Challenges and Quiz together.

# Review: Week 4 Quiz

Sass (.sass) and Scss (.scss) are both valid implementation of the Sass preprocessor language. Sass was the original implementation, and Scss (which stands for Sassy CSS) came after.

Select one:

- True ✓
- False

In jQuery, the code:

**$(function() { ... })**

is a valid and recommended shortcut for the code:

**$(document).ready(function() { ... })**

Select one:

- True ✓
- False

**let pastriesArr = ['muffin', 'cookie', 'pie', 'cupcake', 'strudel'];**

What would you type to retrieve the value of the third element ("pie") in this *pastriesArr* array?

Answer: pastriesArr[2] ✓

# Review: Week 4 Quiz

What is the return value from this function:

```
function applyAdmissionDiscounts(fee, isChild, isMember) {
    if (isChild)
        fee *= .3;
    else if (isMember)
        fee /= 2;
    return fee;
}
```

...if invoked in this way?:

```
applyAdmissionDiscounts(10, true, false);
```

Answer: 3 ✓

# Review: Week 4 Quiz

Which of these are *not* DOM Events? There are TWO answers.

Select one or more:

- [ ] paste
- [ ] dragenter
- [x] onclick ✓ **onclick** is not itself a valid DOM event - it is a keyword used to create an event handler for the event **click**.
- [x] mouseoff ✓ While there are valid mouse events such as **mouseout** and **mouseleave**, **mouseoff** is not a valid event.
- [ ] click

https://www.w3schools.com/jsref/dom_obj_event.asp