Before we begin...

- Open up these slides:
  - https://goo.gl/jJD335

Templating and Lab

Learning Objectives

- **Review** what ECMA Script is, and its relation to JavaScript versioning
- **Identify** another approach to creating strings
- **Explain** interpolation and use it effectively
- **Create** and **use** templates so as to create HTML markup efficiently
- **Identify, explain** and **use** array methods effectively

Agenda

- Review
- Templating
- Lab

A quick review

- JavaScript DOM manipulation patterns
- Events
  - Creating event handlers
  - Using the event parameter
- Timers
- Animations

Templating

ECMAScript?

String Creation

Strings

- You can also create strings with backticks (``)
- This allows for:
  - Multi-line strings
  - Interpolation
- It is a new feature of JavaScript!

Interpolation

- The process of inserting a value into a string
- Almost like substitution
- We can run any JavaScript code with interpolation. For example, call functions

Strings

```
var myString = `Creating strings!`;

var favNumber = 42;

var message = `Favourite Number: ${favNumber}`;

var str = `4 * 2 = ${4 * 2}`;
```

Strings

```
var username = "kookslams";
var postCount = 673;
var description = "Curated kook slammage.";
var followerCount = 1000000;
var followingCount = 348;

var html = `
    <h1>${username}</h1>
    <h3>
        Posts: ${postCount}.
        Followers: ${followerCount}.
        Following: ${followingCount}.
    </h3>
    <p>${description}</p>
`;
```

Functional Array Methods

Imperative vs. Declarative

- An **Imperative** Approach to Programming
  - Describes the "HOW". You explain every single thing in the program
  - e.g. for (...) {}
- A **Declarative** Approach to Programming
  - Describes the "WHAT". You describe a pattern
  - e.g. forEach

Declarative

- Declarative Programming leads to:
  - More readable code
  - Often more efficient code
- You'll spend less time trying to understand your program, and more time figuring out the higher-level logic
- Declarative programming uses the magic and hides the complexity

forEach

forEach

- The .forEach method allows us to iterate through each item in a collection
- We provide a callback function that will be provided with the current item, the current index and the entire collection

forEach

```
ARR.forEach(function (ITEM, INDEX, ARR) {

});
```

```
var letters = ["a", "b", "c", "d", "e"];

letters.forEach(function (letter, index) {
    console.log(
        `Current Letter: ${letter}. Index: ${index}`
    );
});
```

filter

filter

- The .filter method allows us to iterate through each item in a collection
- It will return a new collection
- We provide a callback function that will be provided with the current item, the current index and the entire collection
  - If the callback function returns true, the item will be stored in the returned collection. Otherwise, it won't be. The callback must be a predicate!

filter

```
ARR.filter(function (ITEM, INDEX, ARR) {
    // Must return a boolean!
});
```

```
var numbers = [1, 2, 3, 4, 5, 6];

var evens = numbers.filter(function (num) {
    return num % 2 === 0;
});

console.log(evens);
```

map

map

- The .map method allows us to iterate through each item in a collection
- It will return a new collection
- We provide a callback function that will be provided with the current item, the current index and the entire collection
  - The callback must return a value! The value that you return will be stored in the new collection
  - Essentially it transforms each item!

map

```
ARR.map(function (ITEM, INDEX, ARR) {
    // Must return a boolean!
});
```

```
var letters = ["a", "b", "c", "d", "e"];

var upperCased = letters.map(function (letter) {
    return letter.toUpperCase();
});

console.log(upperCased);
```

map

```
ARR.map(function (ITEM, INDEX, ARR) {
    // Must return a boolean!
});
```

```
var numbers = [1, 2, 3, 4, 5, 6];

var multiplied = numbers.map(function (num) {
    return num * 5;
});

console.log(multiplied);
```

reduce

reduce

- The .map method allows us to iterate through each item in a collection
- It will return a new collection
- We provide a callback function that will be provided with the running total and the current value, as well as a starting value
  - The callback must return a value! The value that you return will be stored as the running total value for the next iteration
  - Often called *inject*

reduce

```
ARR.reduce(function (TOTAL_VALUE, CURRENT_VALUE) {
    // Must return a value! This will be set to TOTAL_VALUE
}, STARTING_VALUE);
```

```
var nums = [1, 2, 3, 4, 5, 6];

var total = nums.reduce(function (sum, currentNumber) {
    return sum + currentNumber;
}, 0);

console.log(total);
```

reduce

```
var brothers = [
    { name: "Groucho" },
    { name: "Harpo" },
    { name: "Zeppo" },
    { name: "Chico" },
    { name: "Gummo" }
];

function createMarkup(allHTML, brother) {
    return allHTML + `<li>${brother.name} Marx</li>`;
}

var htmlMarkup = brothers.reduce(createMarkup, "");

document.body.innerHTML += `<ul>${htmlMarkup}</ul>`;
```

Exercise

Generate some HTML markup from this

Lab

Lab

Homework

- Finish off the lab

Homework

- Dancing Cats!
  - Here is some inspiration

Hopefully we will see some demos of this!

Homework (Extra)

- Go through The Modern JavaScript Tutorial
- Read Eloquent JavaScript
- Read Speaking JavaScript

What's next?

- More JavaScript!

Questions?

Feedback

https://ga.co/js05syd

Thanks!