FRONT-END APPLICATION DEVELOPMENT WITH JAVASCRIPT

PROFESSIONAL & CONTINUING EDUCATION

UNIVERSITY *of* WASHINGTON

CLASS GOALS

DEVELOPERS WILL BE ABLE TO...

> Build single-page applications (SPAs) using ReactJS

> Utilize JavaScript and React design patterns

> Incorporate APIs and routing into a front-end application

> Deploy a front-end application to a hosting provider

> Research and comprehend unfamiliar concepts via reading documentation, asking peers, and asking instructors

WEEK 1

JAVASCRIPT REVIEW + ECMASCRIPT SYNTAX

LEARNING OBJECTIVES

DEVELOPERS WILL BE ABLE TO...

> Understand JavaScript scoping and closure behavior

> Identify and create higher-order functions

> Setup a NodeJS developer environment

> Create JavaScript modules

> Write JavaScript that leverages modern ECMAScript features, such as let/const, arrow functions, classes, template literals, and object/array destructuring

DEVELOPMENT SETUP

- Install Node
  - https://nodejs.org/en/
- Github
- Text Editor (I'll be using Visual Studio Code)
  - https://code.visualstudio.com/
- Terminal (I'll be using iTerm + Oh My Zsh)
  - https://www.iterm2.com/
  - https://ohmyz.sh/

JAVASCRIPT
A REVIEW

- Variable scope
  - Global
  - Function
- Functional programming
  - Higher-order functions

ECMAScript Syntax

GOODBYE VAR
HELLO LET/CONST

> Unlike var, let and const provide the following benefits when declaring variables:
>> Declare **once** and **only once**
>> Limit **scope** to the immediate block

```
var time = '06:00GMT';
```

```
const school = 'UW';

let grade = 'A';
```

GOODBYE VAR
HELLO LET/CONST

```
var school = 'UW';
var school = 'WSU';

console.log(school);
// WSU
// Oh no!
```

```
const school = 'UW';
school = 'WSU';

// TypeError
```

```
let school = 'WSU';
school = 'UW';

console.log(school);
// UW
// Nice 😎
```

A NEW WAY TO FUNCTION
ARROW FUNCTIONS

- Arrow functions are leaner versions of functions
  - Short syntax
  - Inherits "this"
    - Therefore, arrow functions can't be used for a constructor
  - No arguments object or prototype

```
const numbers = [1, 2, 3];
numbers.map(function(num) {
  return num * 2;
});
```

```
const numbers = [1, 2, 3];
numbers.map(num => num * 2);
```

A NEW WAY TO FUNCTION
ARROW FUNCTIONS

```
const numbers = [1, 2, 3];
numbers.map((number) => {
  return number * 2;
});

// parens optional with
// single parameter
numbers.map(number => {
  return number * 2;
});
```

```
const print = (arg) => {
  console.log(arg);
}

function print(arg) {
  console.log(arg);
}

// One-line arrow function
// returns the expression
const sum = (a, b) => a + b;
```

A NEW WAY TO FUNCTION
ARROW FUNCTIONS

```
function Announcer() {
  this.message = 'Time is up';
  setTimeout(function() {
    console.log(this.message);
  }, 1000);
}

new Announcer();
// undefined
```

```
function Announcer() {
  this.message = 'Time is up';
  setTimeout(() => {
    console.log(this.message);
  }, 1000);
}

new Announcer();
// Time is up
```

GOODBYE STRING CONCATENATION HELLO TEMPLATE LITERALS

- Instead of string concatenation, we can embed variables directly into strings
  - Must be defined with backticks (`)
  - Embed variables with syntax:
    - ${myVarName}
  - Supports multiline strings

```
const name = ‘Mary’;
const greeting = ‘Hello ‘ + name;
```

```
const name = ‘Mary’;
const greeting = `Hello ${name}`;
```

TRY IT OUT

- Write a script that generates a “mad lib” using variables and a template literal.
- Should meet the following requirements:
  - Generate a “mad lib” using various parts of speech (noun, verb, adjective, adverb, exclamation, etc.)
  - Parts of speech should be defined as separate variables

Example “Mad Lib”:

__Exclamation__! I never expected a ___Noun___ to show up at my bus stop and start dancing ___Adverb___.

TRY IT OUT – EXAMPLE STARTER CODE

```
const noun = ‘apple’;
const properNoun = ‘Paris’;
const verb = ‘run’;
const adverb = ‘quickly’;

const madLib = ``; // TODO
```

SO CLASSY CLASSES

- Classes are “syntactic sugar” for JavaScript prototypes
  - Behavior is the same, but we can write them in a cleaner and easy-to-understand manner
- Concepts
  - Definition
  - Constructor
  - Instance and static methods
  - Inheritance

SO CLASSY CLASSES

```
class Book {
  constructor(title) {
    this.title = title;
  }
}

// or

const Book = class {
  constructor(title) {
    this.title = title;
  }
}
```

```
class Book {
  constructor(title) {
    this.title = title;
  }

  printTitle() {
    console.log(this.title);
  }

  static sameTitles(b1, b2) {
    return b1.title === b2.title;
  }
}

const book = new Book('Harry Potter')
book.printTitle();
```

SO CLASSY CLASSES

```
class Book {
  constructor(title) {
    this.title = title;
  }
}

class ComicBook extends Book {
  constructor(title, issueNumber) {
    // passed to the parent Book constructor
    super(title);
    // instance assigned to ComicBook (not the parent Book)
    this.issueNumber = issueNumber;
  }
}
```

...

DESTRUCTURING

- Destructuring is an assignment syntax
- Allows the “unpacking” of arrays and objects into variables

```
const person = {
  name: ‘John’
};

const { name } = person;

console.log(name);
// John
```

```
const list = [1, 2, 3, 4];

const [a, b, c, d] = list;

console.log(c);
// 3
console.log(a);
// 1
```

DESTRUCTURING

> When unpacking arrays/objects, values can be ignored or set to a default if no value exists

```
const person = {
  name: ‘John’
};

const {
  name
  age = 99
} = person;

console.log(age);
// 99
```

```
const list = [1, 2, 3];

const [a, b, c, d = 4] = list;

console.log(d);
// 4
```

REST/SPREAD OPERATOR

> Using the rest/spread operator (...), properties that aren't handled using destructuring are either collected together or spread apart.

```
const numbers = [9, 3, 1, 6];
Math.min.apply(null, numbers);
// 1
```

```
const numbers = [9, 3, 1, 6];
Math.min(...numbers);
// 1
```

```
const [first, ...theRest] = [9, 3, 1, 6];
console.log(...theRest);
// 3 1 6
```

TRY IT OUT

> Design a class that creates an object from these lines of code:

```
// Expense info (name of expense, price before tax, tax)
const printerExpense = ['Printer toner', 50, 5];

const expense = new Expense(...printerExpense);
```

> Add an instance method to the class that will calculate and print the total cost of the line item.

DEFAULT PARAMETERS

- When destructuring an object or array, we could set default values if the value was undefined.
- We can do the same with function parameters

```
function calculateThis(x, y, z = 3) {
  return (x + y) * z;
}

calculateThis(2, 3)
// 15
```

ENHANCED OBJECT PROPERTIES

> Property definition shorthand

```
const value = 4.00;
const currency = ‘USD’;

const amount = { value, currency };

// Same as
// const amount = {
//   value: value,
//   currency: currency
// };
```

MODULES

> A couple module patterns
>> CommonJS (which you may see in older code)
>> ES Modules (which you'll use in this class)

```
const Car = require('./Car');

class Truck extends Car {}

module.exports = {
  Car,
  Truck
};
```

```
import Car from './Car';

class Truck extends Car {}

export {
  Car,
  Truck
};
```

MODULES

> More ES Module examples

```
import Car from ‘./Car’;

import * as Car from ‘./Car’;

import {
  Car, Jeep
} from ‘./Car’;

import {
  drive as driveMyCar
} from ‘./Car’;

import {
  default as MyCar
} from ‘./Car’;
```

```
export default class Car {};

export function drive() {};

export const mph = 45;

export {
 Car,
 Truck,
 SUV,
 Jeep
};
```

MODULES

- Node + Browser support for ES Modules still varies, so in order to use these modules, we'll be using
  - Babel
  - create-react-app (which uses Babel under the hood)
- https://babeljs.io/

Always important to note that many ECMAScript have been around for a while, but support will vary across different browsers and environments.

Check your environment to make sure it supports the features you're using or compiles your code to something compatible with most environments.

BUT WAIT, THERE’S MORE

We covered common ECMAScript features that appear in ReactJS. We’ll bring up more throughout the class. Here’s a more extensive list of other features:

https://en.wikipedia.org/wiki/ECMAScript

FOR NEXT WEEK

Assignment + React tutorial

> Before you leave today, install create-react-app using npm

  > `npm install -g create-react-app`

  > `create-react-app ./test-app`

> Complete Week 1 assignment

> Complete a React tutorial

  > https://reactjs.org/tutorial/tutorial.html

  > Optionally, the step by step guide: https://reactjs.org/docs/hello-world.html

> Reach out on Slack if there are any questions or setup issues.