# Prep:  Dive into React

Below you will find a short video about React. Watch the video then answer the questions below.

A High Level Overview Of React

1. What is React?
2. What is a component?
3. What is the dataflow of React?
4. How do we make a React element a DOM element?
5. React is a User Interface _____.
6. Which direction does data flow in React?
7. Every component manages its own ____.

# Computer Setup

Confirm that your laptop or computer is properly setup with appropriate software and settings so that you may begin your class work.

1. Ensure your computer is compliant with Code 201 Requirements 1.Install and configure the required Code 301 Developer Tools

## Submission Instructions

Submit a screen shot of your terminal with the success messages from each verification step above.

# Create a Reading Notes Repo

If you don't already have one, create a `reading-notes` repository on GitHub, where you can keep track of your observations and questions from the reading assignments throughout your course.

Think of it as a way to document and highlight your new knowledge. Publish it with GitHub pages, so it becomes a live web site.

## Setup and Documentation

Use these resources to get going:

- Markdown on GitHub
- Getting Started with GitHub Pages

## Requirements

1. If you have previously created this repo for another class, double-check to ensure it follows all guidelines here.
2. Your site should exist at `https://USERNAME.github.io/reading-notes/`
3. The "source" for your site should be a README.md file, written with Markdown
4. On the main page:
    - Include a level-1 heading titled `Reading Notes`
    - Include a description of what this web site is about
    - Include a level-2 heading for each course, ie:
        - `Code 102 - Intro to Software Development`
        - `Code 201 - Foundations of Software Development`
        - `Code 301 - Intermediate Software Development`
        - `Code 401 - Advanced Software Development`
5. Utilize at least 5 different features of Markdown to structure your page

### Submission Instructions

Submit the link to your reading notes repo, even if you have already done so for a previous class. Your grader will verify it is set up correctly for this course.

# CSS Diner

**If you are coming from a Code 201 class that ended within 2 weeks of the start of your Code 301 class**, consider redoing the assignment.

If you completed Code 201 more than 2 weeks prior to the start of Code 301, please complete this assignment as a way to keep your HTML/CSS skills sharp.

If you are testing in to Code 301, you should know that students in Code 201 complete all levels of the CSS Diner. This is your opportunity to take on that challenge.

CSS Diner takes you through different levels of CSS challenges. Once you have completed all the levels, take a screenshot that shows all the levels checked off and submit it.

# Chocolate Pizza CSS

**If you are coming from a Code 201 class that ended within 2 weeks of the start of your Code 301 class**, submit the work you did there.

If you completed Code 201 more than 2 weeks prior to the start of Code 301, please complete this assignment as a way to keep your HTML/CSS skills sharp.

If you are testing in to Code 301, you should know that students in Code 201 complete a time-boxed "design comp" assignment, called "Chocolate Pizza". This is your opportunity to take on that challenge:

- Rather than the “complete as much of the assignment as you can in the available time” approach in Code 201, your challenge is to get the HTML/CSS mockup to be pixel-perfect
- Your column of content should be centered in the window, as in the preview image
- Your solution does not need to be responsive in any way
- Do not use any negative margins, grids, or flex positioning

**Instructions:**

1. Create and clone a new GitHub repository to house the code for this project.
2. Immediately check out a new branch in which to do your work.
3. Download and unzip these assets into an `assets` directory in your project.
4. Write your code in two files: `index.html` and `style.css`. You do not need to include JavaScript.
5. Use the included `PREVIEW.png` image as a reference; your goal is to match it as closely as possible.
6. Make regular Git commits with appropriately descriptive commit messages while you are working.
7. When finished, be sure to push your final code to GitHub and merge your branch into `main`.
8. Deploy your page on GitHub Pages via the options in the repository “Settings” tab.
9. Submit the links to your repository AND your deployed page in the corresponding Canvas assignment.

# Portfolio Prep

In the first module of this course, you will build a personal portfolio site. This will help you establish a home on the web, and show off some of your software development projects and skills.

Sometimes, the hardest part is getting the words right. To help you prepare for building your portfolio, you can start thinking about what you want to say now.

To find some inspiration, check out these snapshots of a student’s completed portfolio:

- Home Page
- About Page

## Tell me about yourself

Visitors to your portfolio site want to learn about you, in memorable, bite-sized bits. Your page will have a space for you to fill in each of the following descriptions of yourself:

1. A two or three word catchy title. Do NOT use cliches like “programming ninja” or “coding rockstar”.
2. A personal headline, like you have atop your LinkedIn page. What do you want your career to be about?
3. Your professional pitch: You’ve done a recording, so just write down here how it goes.

4. What excites you the most about tech? Write 1-2 sentences.

## Gather your assets

A few punchy images will really help your page stand out, and make it your own. Gather images for the following:

1. A headshot of your wonderful face, approximately 200x250 pixels.
2. A background image, approximately 1500x700 pixels.
3. At least three screenshots of the best-looking parts of a previous project, like your Code 201 final project.
4. At least three screenshots each, of two other projects you have worked on. Salmon Cookies? Odd Duck?

## Submission

1. Create a new repository on GitHub, called `portfolio-prep`.
2. In this repository:
    - Use the `README.md` to organize and collate your information and notes.
    - Upload the images that you've sourced for the portfolio.
    - Submit a link to this repository.

# Readings and Resources: ES6 Classes

Below you will find some reading material, code samples, and some additional resources that will help you to better understand ES6 Classes

- Watch the
- Shred Talk: ES6 Classes
- Links to an external site.

- 
- Review the Demo Code
- Links to an external site.
- 
- Complete the Assignment

# Objects and Inheritance

JavaScript objects use prototype-based inheritance. Its design is logically similar (but different in implementation) from class inheritance in strictly Object Oriented Programming languages like Java and C#.

It can be loosely described by saying that when methods or properties are attached to object's prototype they become available for use on that object and its descendants, but not directly attached to them.

When you use class and extends keywords internally JavaScript will still use prototype-based inheritance. It just simplifies the syntax (this is often called "Syntactic Sugar"). While classes are easier to use, it's still very important to understand how prototype-based inheritance works. It's still at the core of the language design.

## Prototypal Inheritance

```
Unset
function Animal(name) {
  this.name = name;
}
Animal.prototype.walk = function() {}

function Bird(name) {
  // I can do all the things animals can do!
  Animal.call(this, name);
}
```

```
// Unlike other animals, birds can fly
Bird.prototype.fly = function() {}

// Make a new bird ..
let parrot = new Bird('parrot');
parrot.fly();
parrot.walk();
```

## ES6 Classes

The same thing with classes (much cleaner!)

- `function()` becomes `class {}`
- `call()` becomes `extends`
- Classes are standalone, self-contained object (instance) factories
    - Ultimately, they result in a prototype
    - But for the developer, they are many orders of magnitude easier to comprehend and work with

Unset

```
class Animal {
  constructor(name) {
    this.name  = name;
  }

  walk() {}
}

// Thanks to 'extends', all birds can now do all things
animals can
class Bird extends Animal {
  // Birds can walk, becuase they're animals also do their
own thing.
  fly() {}
```

```
}

// Make one (the same was as before, but wasn't the class
creation much easier?)
let parrot = new Bird('parrot');
parrot.fly();
parrot.walk();
```

# Additional Resources

- Video:
- what the heck is the event loop anyway
- Links to an external site.

-
- MDN inheritance
- Links to an external site.
-
- MDN this
- Links to an external site.
-
- MDN class
- Links to an external site.
-

# ES6 Arrow Functions

This assignment is designed to introduce you to some features in ECMAScript 2015

Links to an external site.

, otherwise known as ES6. From this point on, you are expected to use these features.

# Overview: Arrow Functions

- MDN docs
- Links to an external site.
-
- caniuse.com
- Links to an external site.
-

By now you should be comfortable writing function expressions

Links to an external site.

and function declarations

Links to an external site.

. Arrow functions are a shorter, more concise way to write a JavaScript function. The syntax might seem strange at first, so try to use arrow functions wherever you can and they will become second nature in no time.

There are a few caveats with arrow functions, though. Most importantly, the `this` context is not reset within an arrow function. The value of `this` is therefore the same as the `this` of the enclosing scope (the surrounding non-arrow function). If there isn't a non-arrow function scope surrounding, the `this` context will be, in the browser, the global window object.

Why does this happen? It happens because arrow functions retain the `this` value of the enclosing functional scope. Therefore, you will want to avoid using an arrow function in a constructor (where we need the contextual `this` to be the object we are building) or any method that needs to use `this` to behave properly.

# Assignment Tasks

1. Create a new repo in your GitHub account called `arrow-functions`. Clone the empty repo to your dev environment.
2. Navigate to the class repo
3. Links to an external site.
4. Then navigate into the `arrowfunctions` folder.

5. Copy the contents of the folder named `starter-code` into your newly-created repo. Make an initial commit with the unchanged starter code.
6. Proceed to work on a well-named branch. As you work, remember to add, commit, and push regularly.
7. The `app.js` file contains examples of function expressions, as you are accustomed to seeing. Work through steps 1-9, reading the notes and reviewing the refactored samples.
8. For each of these steps, uncomment the console.log line. Open the `index.html` file in the browser and verify the correct output in the developer console.
9. To complete step 10, refactor the function expressions one at a time. Uncomment the console.log line and use it to check that the output is the same after you have completed the refactoring process.
10. To complete step 11, uncomment the two console.log lines and observe the output in the developer console in the browser. Answer the corresponding questions.

# Additional resources

- "JavaScript Arrow Functions" by Wes Bos
- Links to an external site.
-

# Submission Instructions

1. When finished, make a PR from your branch to main, and merge it.
2. Submit a link to your PR. You can link to a pull request even if the pull request is already merged or closed.
3. Add a comment in your Canvas assignment which includes the following:
    - A question within the context of today's lab assignment
    - An observation about the lab assignment, or related 'Ah-hah!' moment
    - How long you spent working on this assignment