Review Questions

1. **What is nesting in code and why do we do it? What are some examples of nesting?**

When elements go inside other elements, use it as a hierarchy so the browser knows what the parent and children are.

2. **What are some common content tags we use in HTML?**

div, p, ol, ul, li, a.

3. **How do we write HTML comments?**

<!-- Comment -->

4. **What is a CSS comment?**

/\* Comment \*/

5. **What are CSS declarations? Write a declaration on the board. Can they be grouped together? What must they be surrounded by? What do they end with?**

CSS declarations are made up of the property and value you want to apply. Ex. background-color: red. Yes they can be grouped together. They have to be surrounded by curly brackets. Each declaration has to end with a semicolon.

6. **What is a CSS selector?**

A selector instructs the browser to search the page for any HTML element that matches the criteria. Ex. class, id, element name.

7. **Draw three different examples of CSS selectors.**

p, .paragraph, #paragraph.

8. **What are some examples of CSS properties?**

border-radius, width, height, etc.

9. **What are 3 different ways to define colors in CSS?**

rgba, hex, color name.

10. **What are some different CSS unit values?**

pixels, rem, em, percentages.

11. **What is the difference between block and inline elements?**

Block elements take up the full space of the container, inline only take up size of the content.

12. **What are some default block elements?**

div, p, li, ul

13. **What are some default inline elements?**

img, span, a

14. **What does inline-block do?**

You can declare and width and a height to the elements.

15. **What is the box model? Can you draw an example of it?**

Idea that every element is the total of it’s content, padding, border, margin.

16. **What is padding? What is margin? What is border?**

17. **What are the different ways you can write the padding, margin, and border declarations?**

top, bottom, left, and right or all in one line

18. **What is a way we can center an element with margin?**

margin: 0 auto.

19. **What are the different types of positioning we use in CSS? What do each do?**

relative, absolute, fixed. Relative stays in page flow but can be manipulated, absolute will be moved based on relative container, fixed will stay relative to browser window.

20. **What does floating an element mean?**

Takes an element from normal flow and pushes it to the right or left of it’s parent element.

21. **What do you always need to declare with float?**

width, if it’s left or right.

22. **What is the clearfix and why do we use it?**

What *clearfix* does is force content after the floats or the container containing the floats to render below it.

23. **What is the z-index? What does it only work with? If one element had a z-index of one and the other had two, which would be the top layer?**

z-index defines the order of overlapping elements, they need to be used with positioning. The highest number goes on top so the number two element would be on top.

24. **What are some CSS pseudo classes?**

a:visited, a:hover.

25. **What is the nth-child selector? What would it look like if I selected the third child of an element? What if I wanted to select every fourth item?**

It takes a mathematical expression to determine which child to select. :nth-child(3), :nth-child(4n or 4n + 4)

26. **What is responsive design? How do we make a site responsive with CSS?**

Responsive design is making all sites look good on all screens and devices. We use media queries.

JavaScript questions

27. **What are two ways we output information with JS? Describe both and why they are used.**

console.log and return. Console log is used for debugging and prints things to the console, the return statement ends function execution and specifies a value to be returned to the function caller.

28. **What are some keywords in JavaScript?**

var, if, else if, for, while, do while, function, return.

29. **How do we write comments in JavaScript?**

// This is a comment, you can also do /\* This for a comment \*/

30. **What are variables? What are ways we declare variables?**

Containers we store our data in. We declare variables by using the keyword var and then the name we want to use for the variable.

31. **How do we initialize a variable?**

Giving it a value. var x = something.

32. **What are the six different primitive data types? What do they look like?**

null, number, boolean, undefined, symbol, string are all acceptable answers. Objects are the other data type. Arrays and functions are objects. If they ask what a symbol is:

Number = numeric data type, string = sequence of characters that represent text, boolean = true or false, null = non-existent value, undefined = is a value that is given, by default, to variables that have been declared yet not initialized, objects = collection of properties and values.

~ Symbols are just a slightly different way to attach properties to an Object - you could easily provide the well-known symbols as standard methods, just like Object.prototype.hasOwnProperty which appears in everything that inherits from Object (which is basically everything).

33. **What are truthy and falsy values? Some examples of falsy? Some examples of truthy?**

Any value in JS can be treated as true or false, falsy treated as if false. Some examples of falsy are false, null, 0, undefined, NaN, empty strings. Some examples of truthy are true, non-zero numbers, non-empty strings, mathematical expressions.

34. **What are some examples of arithmetic operators?**

+, -, \*, /, %, ++, --

35. **What are ways we compare things in JS? Which ones are strict? Which are shallow? What’s the difference between them?**

===, !==, >, <, >=, <=, strict is ===, shallow is ==, shallow does type coercion.

36. **What are the 3 logical operators in JS?**

&&, ||, !

37. **How do we write an if statement? If - else? If - else if - if?**

if (condition) {

// code

} else if (condition) {

// code

} else {

// code

}

38. **Create a variable declaration called numberOfSeats and initialize it with the value of 6. If the number of seats are greater than 6, console.log that they get a discount, if not, console.log they don’t get a discount.**

var numberOfSeats = 6;

if(numberOfSeats > 6) {

console.log(“You get a discount!)

} else {

console.log(“You don’t get a discount!);

}

39. **What are the 3 different types of loops?**

for loop, while loop, do while.

40. **How is a for loop written? What does each part of the for loop mean?**

for (var i = 0; i < something; i++) {

//code runs here

}

Initialization, condition, final expression

41. **Write a for loop that logs all even numbers between 40 and 100.**

for (var i = 40; i <= 100; i ++) {

if (i % 2 === 0) {

console.log(i);

}

}

42. **How do we write while loops? Do while loops?**

var i = 0

while (i < 20) {

i++;

console.log(i);

}

do {

i++;

} while (i < 20)

43. **What are functions? Why do we use them?**

Functions are the verbs of Javascript, they allow us to reuse our code over and over again.

44. **What does the basic structure of a function look like?**

function nameOfFunction() {

//things the function does

}

nameOfFunction();

45. **What are arguments? Where do they go in the function? What are parameters? Where do they go in the function?**

Arguments go in the function call, assign values to the parameters. Parameters are names assigned to the data referenced within a function.

46. **Declare and initialize two variables named userName and userHobby. Create a function named intro with two parameters named name and hobby. Pass in the variables as the two arguments when the function is called.**

var userName = “Yasmine”;

var userHobby = “Reading”;

function intro(name, hobby) {

console.log(“Hello my name is “ + name + “ and my hobby is “ + hobby);

}

intro(userName, userHobby);

47. **What is an array? Draw one for us. What types of data can be stored inside an array?**

In JavaScript, an is a data structure consisting of a collection of elements (values or variables), each identified by a numerical index. This numerical index starts at 0, not 1. Anything that can be stored in a variable can be stored in an array.

48. **What does zero base indexing mean for arrays? What is the index of the third item in an array?**

It starts with 0 therefore index of the third item would be two.

49. **What is .length? In var array = [1, 2, 3, 4] what would array.length be?**

.length tells us the number of items. In this example it would be 4. The property returns the length of an object. E.g., if the object is an array, the number of elements is returned; if a string, the number of characters.

50. **What is bracket notation? How can we use it to assign new elements into an array?**

Bracket notation can be used to access, reassign elements in an array, or add new elements.

var drinks = [“Coke”, “Coffee”, “Sweet tea”];  
drinks[0] = “Water”; // [“Water”, “Coffee”, “Sweet tea”]  
drinks[4] = “Green tea”;

51. **What are objects? How do we write them?**

Objects are a data structure that allow us to store collections of data. They are very good for bundling data together for more complex programs.Objects contain properties and values.  
Objects require curly braces(unlike arrays, which use square brackets). We can populate them with properties and values, separated by a colon. If there are multiple properties, we use a comma to separate them.

var currentUser = {

name: “Yasmine’',

hobby: “Reading”

}

-Get the value of the hobby property from the object.

console.log(currentUser.hobby);

52. **What are two ways object properties can be accessed?**

Dot notation and bracket notation.

53. **What does each way look like?**

Dot notation = object.property, bracket notation = object[“property”]

You can reassign values or create new values using both ways.

54. **How do you delete a property from an object?**

delete object.property

55. **What are methods? What are some methods that we’ve seen before.**

They allow us to do something to an object. Any built in method works toUpperCase(), toLowerCase(). They change or return something from the object.

56. **What does the method do we use to convert data types into a string?**

.toString()

57. **What does charAt() do?**

Returns the character at a specified index of a string.

var favFood = "Pineapple"; console.log(favFood.charAt(0)); // "P"

58. **What does concat() do?**

The method is used to combine multiple strings into one.  
var favFood = "My favorite food is pineapple.";  
var favDrink = " My favorite drink is coffee";  
var favMeal = favFood.concat(favDrink);  
console.log(favMeal); // "My favorite food is pineapple. My favorite drink is coffee"

59. **What does indexOf() do?**

The method returns the rst index at which the speci ed value occurs.  
var favFood = "My favorite food is pineapple.";  
var favDrinks = [“coffee”, “water”, “pineapple juice”, “orange juice”];

favFood.indexOf("f"); // > 3  
favDrinks.indexOf("water"); // > 1

60. **What does pop() do?**

The method removes the last element in an array and returns it.  
var foodList = [“pizza”, “donuts”, “cookies”, “salad”];

foodList.pop(); // "salad"  
console.log(foodList); // ["pizza", "donuts", "cookies"]

61. **What does push() do?**

The method adds one or more elements to the end of an array and returns the updated length of the array.  
var foodList = [“pizza”, “donuts”, “cookies”, “salad”];

foodList.push(“eggs”);  
console.log(foodList); // ["pizza", "donuts", "cookies", "salad", "eggs"]

62. **What does shift() do?**

The method removes the rst element in an array and returns it.  
var foodList = [“pizza”, “donuts”, “cookies”, “salad”];

foodList.shift(); // "pizza"  
console.log(foodList); // ["donuts", "cookies", "salad"]

63. **What does unshift() do?**

The method adds one or more elements to the beginning of an array and returns the updated length of the array.  
var foodList = [“pizza”, “donuts”, “cookies”, “salad”];

foodList.unshift(“pineapple”);  
console.log(foodList); // ["pineapple", "pizza", "donuts", "cookies", "salad"]

64. **What is the forEach method?**

The method accepts a function as an argument. That functions accepts an element from the array as an argument. Then the body of that function is executed for each (see what they did there?) element in the array. For each is an even safer option for iterating over an array than a regular for loop.  
var userList = [“Adam”, “David”, “Yasmine”];

userList.forEach(function(name) {  
 console.log(name)

});

65. **Declare a variable called bands, initialize it as an array of strings with 4 values. Use the forEach method to log each element of the array to the console. Add the string ‘are cool’ to each item.**

var bands = [“kiss”, “rolling stones”, “AFI”, “muse”];

bands.forEach(function(items) {

console.log(items + “ are cool”);

})

66. **Declare a variable called addressBook, and initialize it as an array of objects. Give 3 each object two properties called name and address and assign them each values. Use a forEach to loop through them and print them to the console.**

var addressBook = [  
 {  
 name: “Yasmine”,  
 address: “123 Fake Street”  
 },  
 {  
 name: “Faraz”,  
 address: “456 Another Street”  
 },  
 {  
 name: “Mom”,  
 address: “444 Road”  
 }  
]  
  
addressBook.forEach(function(items) {  
 console.log(items.name + " : " + items.address);  
});

67. **What is the DOM?**

The document object model (DOM) is an interface which allows programs and scripts to dynamically access and update the content, style and structure of an HTML document.

68. **What are some ways we can access DOM objects?**

You can use nodes, but that’s considered tedious. Better to use methods. getElementsByTagName, getElementsByClassName, getElementById, querySelectorAll

69. **Which will return arrays? What if you want a specific object in the array?**

getElementsByTagName and getElementsByClassName, querySelectorAll use bracket notation.

70. **How do we create an element in the DOM?**

In order to create an element, we must first create a variable to store this object in.  
var newElement = document.createElement(“element”);

71. **How do we then place our new element in the DOM?**

document.body.appendChild(newElement);

72. **How do we change the text of an element in the DOM?**

var newElement = document.createElement(“div”);  
newElement.innerText = "Yoooooo!";  
  
document.body.appendChild(newElement);

73. **What does an event refer to on the DOM?**

Many user interactions, or events are registered by the browser.  
We can write code that triggers or runs, when a certain event is registered.

74. **What are some examples of events we’d use on the DOM?**

Scroll, click, resize, keyup, mouseover.

Week 4 Assessment Review Questions

**1. What is an IIFE and why do we use it?**

An IIFE is an anonymous function that is invoked as soon as it's created. It creates a closure that keeps all the variables inside private and contained.

It is used to keep the global namespace from being cluttered with all of the variables and functions in a javascript le. When you use an IIFE, all of its variables are private because they are not visible out of the IIFEs scope.

jQuery and AJAX

1. **What are some reasons people use jQuery over vanilla javascript?**

Small, fast, feature-rich, cross-browser, extensively documented, extremely popular

**2. How do we write jQuery selectors? For a class? For an id? For a tag?**

$(‘.class’), $(‘#id’), $(‘tagname’)

**3. Rewrite this into jQuery:**

1. **document.getElementsByClassName(‘nav-links’);**
2. **document.getElementById(‘main-link’);**
3. **document.getElementsByTagName(‘h1’);**
4. $(“.nav-links)
5. $(“#main-link)
6. $(“h1”)

**4. Use the on method to attach a click event to the element with the id of main-link so that it adds and removes the class “is-active” when clicked on.**

$(“h1”).on(“click”, function(){

$(this).toggleClass(“is-active”);

}

**5. What are some other jQuery methods/effects you can use? What do they do?**

.slideUp, .slideDown, slideToggle, hide, show, fadeIn, fadeToggle (make sure they know the slide method)

**6. What’s the difference between GET and POST?**

**\***\*\*GET\*\*\* - Requests data from a specified resource  
\*\*\*POST\*\*\* - Submits data to be processed to a specified resource  
GET is basically used for just getting (retrieving) some data from the server. Note: The GET method may return cached data.  
  
POST can also be used to get some data from the server. However, the POST method NEVER caches data, and is often used to send data along with the request.

**7. Write a GET request from** [**www.something.com/items.json**](http://www.something.com/items.json)**. Use a parameter called data.**

$.get(‘[www.something.com/items.json](http://www.something.com/items.json)’, function(data){

})

**8. Inside the get request, initialize a variable named items that stores the data from the url.**

$.get(‘[www.something.com/items.json](http://www.something.com/items.json)’, function(data){

var items = data;

})

**9. Console.log items.**

$.get(‘[www.something.com/items.json](http://www.something.com/items.json)’, function(data){

var items = data;

console.log(items);

})

**10. Let’s say inside the data there was an array of objects called names, and a property inside the objects called firstName. How would we log the first names of the first index from the array of objects?**

$.get(‘[www.something.com/items.json](http://www.something.com/items.json)’, function(data){

var items = data;

console.log(items.names[0].firstName);

})

Angular

**1. What is Angular?**

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.  
  
AngularJS is what HTML would have been, had it been designed for applications. HTML is a great declarative language for static documents. It does not contain much in the way of creating applications, and as a result building web applications is an exercise in what do I have to do to trick the browser into doing what I want?  
  
The impedance mismatch between dynamic applications and static documents is often solved with:  
  
a library - a collection of functions which are useful when writing web apps. Your code is in charge and it calls into the library when it sees fit. E.g., jQuery.  
frameworks - a particular implementation of a web application, where your code fills in the details. The framework is in charge and it calls into your code when it needs something app specific. E.g., durandal, ember, etc.

AngularJS takes another approach. It attempts to minimize the impedance mismatch between document centric HTML and what an application needs by creating new HTML constructs. AngularJS teaches the browser new syntax through a construct we call directives. Examples include:  
  
Data binding, as in {{}}.  
DOM control structures for repeating, showing and hiding DOM fragments.  
Support for forms and form validation.  
Attaching new behavior to DOM elements, such as DOM event handling.  
Grouping of HTML into reusable components.

**2. What is a Single Page Application?**

In general, a SPA can be characterized by having an outer 'shell' that serves as the header and navigation for the site while the content of the page changes as different parts of the site are visited.

**3. What does MVC stand for? What does it mean?**

The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller.

Model: the data  
View: what the user sees  
Controller: the logic that brings it all together

**4. What are directives in Angular? What are some examples of built in Angular directives?**

Directives are where you manipulate the DOM and catch DOM events. This is why the directive's compile and link functions both receive the "element" as an argument. You can  
  
-define a bunch of HTML (i.e., a template) to replace the directive  
-bind events to this element (or its children)  
-add/remove a class  
-change the text() value  
-watch for changes to attributes defined in the same element (actually it is the attributes' values -that are watched -- these are scope properties, hence the directive watches the "model" for changes)

Examples of built in directives are: ng-app, ng-model, ng-class, etc.

**5. What does the directive ng-model do?**

The ngModel directive binds an input,select, textarea (or custom form control) to a property on the scope using NgModelController, which is created and exposed by this directive.

ngModel is responsible for:  
  
-Binding the view into the model, which other directives such as input, textarea or select require.  
Providing validation behavior (i.e. required, number, email, url).  
-Keeping the state of the control (valid/invalid, dirty/pristine, touched/untouched, validation errors).  
-Setting related css classes on the element (ng-valid, ng-invalid, ng-dirty, ng-pristine, ng-touched, ng-untouched, ng-empty, ng-not-empty) including animations.  
-Registering the control with its parent form.

ngModel will try to bind to the property given by evaluating the expression on the current scope. If the property doesn't already exist on this scope, it will be created implicitly and added to the scope

* **Using the following HTML, describe what the filter is doing.**

<p ng-repeat=”dog in dogs | orderBy”> {{ dog }} </p>

* **Based on the following HTML, describe what is wrong.**

<li ng-repeat=”person in people”> {{ people }} </li>

**6. Create an text input and use ng-model to bind the property ‘name’ to the scope object.**

<input type = “text” ng-model=”name”>

**7. What does the directive ng-app do?**

Declares an element and all its children as an angular app.

**8. What is a custom directive?**

Directives we can make ourselves to attach desired behavior(or functionality) to components or HTML.

**9. What is a component?**

Used for creating custom HTML elements.

**10. What is the module?**

A container for the different parts of an app including controllers, services, filters, directives. Modules are the building blocks that apps are made of. A module's primary job is to serve as place for the pieces of an app to be registered.

**11. Let’s call our module, “someApp”. How do we create the module with no dependencies in Javascript? How do we link it on the html element?**

angular.module(‘someApp’, []); , <html ng-app=’someApp’></html>

**12. What is a controller?**

Controllers have one job and one job only: to control data moving between the view and the model. They should never do any heavy data processing or logic. They should contain minimal amounts of code. They simply take information from one piece of the application and sends it to another piece. They should never contain logic that will manipulate the DOM.

**13. What does the scope refer to in Angular?**

Scope is Angular's 'glue object' that marries the variables and properties on a controller to the view.

**14. Let’s write a controller called “SignOffController”.**

(function(){

function SignOffController(){

}

angular

.module(“app”)

.controller(“SignOffController”, SignOffController)

})();

**15. Let’s add a property to the scope within the controller called “message” and give it the value, “Goodbye!” inside it.**

(function(){

function SignOffController(){

var vm = this;

vm.message = “Goodbye!”;

}

angular

.module(“app”)

.controller(“SignOffController”, SignOffController)

})();

**16. What does dependency injection refer to?**

Dependency Injection is a concept in software design that allows for the components of a software project to be loosely coupled. This makes them easier to test and change without a ecting the other modules that depend on them.  
  
In Angular, software components (modules, services, and directives) are injected by passing them into the constructor function of whatever it is you're instantiating.

**17. What Angular component do we use to share data between controllers?**

Service. Services are reusable components that are separated from views and can be used across multiple controllers and views. You do this by using the .service() or .factory() methods.

**18. What are words we use to define services in Angular?**

Service, factory, provider

**19. Write an example of a service called ‘MyService’ with a function ‘message’.**

function MyService() {

this.message = function() {}

};  
angular  
 .module("app")

.service("MyService", MyService);

**20. Write a controller called ‘MyController’ using ‘MyService’ we just created.**

function MyController(MyService){

MyService.message();

}

angular  
 .module("app")  
 .controller("MyController", MyController);

**21. What is a factory?**

It’s another way to create a service. With a factory, you create an object, add properties, and then return that object literal. Once injected into the controller, the properties of that object are accessible to the controller.

**22. Write a factory for ‘MyFactory’ with the function ‘MyFunction’.**

function MyFactory() {

return {

MyFunction : MyFunction  
}

function MyFunction() {

}

} // awesome code

angular  
.module("app")

.factory("MyFactory", MyFactory);

Note: We use it in the controller the same way, just create it differently.

**23. What are some built in services we use in Angular?**

$http, $timeout, $location.

**24. What is routing in Angular?**

In a multi-page site we can link to different pages using the same anchor tags and attributes. For a single page application, we need a way to tell the browser to load different content. We can do this with routing.  
  
Originally routing was built-in to Angular. Later, it was decided that it should be maintained as its own module and repository. So we must link to it and include it as a module dependency in order to use it.

**25. How do we configure a route?**

For our routes, we will add a object to our module. Inside this config object we will inject the  
to define our routes and define which controllers are used with what views.  
The function takes a route name as a string and an object with the route's properties such as  
and templateUrl.  
The function defines what the router should do for unknown routes.

angular  
 .module("app", ["ngRoute"])

.config(function($routeProvider) {

$routeProvider

.when("/home", {

templateUrl: "home.html"

}).when("/greatness", {  
 controller: "GreatnessController as routing",

templateUrl: "greatness.html"  
}).when("/amazing", {  
 controller: "AmazingController as routing",

templateUrl: "amazing.html"  
}).otherwise({ redirectTo: "/home" });

});

**26. Let’s configure a route with the path /about to the templateUrl about.html and use the controller aboutCtrl.**

angular  
 .module("app", ["ngRoute"])

.config(function($routeProvider) {

$routeProvider

.when("/about", {

controller: “AboutController as routing”,

templateUrl: "about.html"

\*\*\*\*CAN ALSO DO - template: “<component></component>” AND DON’T NEED A CONTROLLER\*\*\*\*

});

**27. What is a view and how do we use it with routing?**

A view is the visible part of the website (the DOM). When using a router like config , views are usually partial (incomplete) snippets of HTML that are injected into the viewport as needed.

We can link to these snippets of html in our router configuration using the property. We can also define templates inline using the property but this is usually not recommended for anything but the simplest templates.

**28. What’s the difference between template and templateUrl?**

Template is a string of HTML rendered within a directive, templateUrl is a path to an .html file that will render into the directive.

**29. How do we write out custom directives?**

function helloWorld() {

return {  
 restrict: "A",  
 link: function($scope, $element, $attrs) {  
 $element.on("mouseover", function() {

alert("Hello World!");  
});

});

angular

.module(“app”);

.directive("helloWorld", helloWorld);

**30. How do we link it in our HTML?**

<hello-world></hello-world>

**31. Write a component called newHeader to create a custom header that says “Hello World!”**

var newHeader = {

controller: "TestController",

template: "<h1> {{ $ctrl.header }} </h1>"  
}  
function TestController() {

var vm = this;

vm.message = "Hello World!";  
}  
angular  
.module("app")  
.component(“newHeader", newHeader);

**32. Write a GET request in Angular using $http service using the url www.something.com/something.json.**

$http({

url: ‘[www.something.com/something.json](http://www.something.com/something.json)’,

method: ‘GET’

})

**33. Console.log the response:**

$http({

url: ‘[www.something.com/something.json](http://www.something.com/something.json)’,

method: ‘GET’

})**.**then(function(response){

console.log(response);

})