2. How an Angular App gets Loaded and Started

1. 
2. Index.html is served to the client.
   1. This file contains the **selector <app-root></app-root>** of the component which is **bootstrapped**. This tag’s content is replaced with component’s template.
3. 
   1. app.module.ts contains module where we tells about what components to be bootstrapped
4. **ng serve command**: **Rebuilds**, creates **bundles**, write **import statements** in index.html.

3. Components are Important!

1. **Component** = Template + Style + Business Logic.
2. Ng serve command adds bundle script tags 🡺 which starts the angular app which gets info about the app module which gives info about the component to be bootstrapped. Then angular is able to replace the content of <app-root> in index.html with the content of AppComponent’s template.
3. **Component** **Benefits**: Reusable, Maintainable.

4. Creating a New Component

1. **server.component.ts**: **server**: name of the component. **component**: What is this file for (component, directive, service). **ts**: In which language it’s written.
2. **@Component**: Typescript feature to enhance a class.
   1. **Properties: selector:string(id, class, attribute), templateUrl:String(relative path to the template file)**

5. Understanding the Role of AppModule and Component Declaration

1. Every new component/directive/pipe is configured in **app.module.ts**. Angular packages all the components belong to a particular module into a package.   
   Each angular pro has at least one module.
2. **@ngModule**: To convert typescript class into module.
   1. **Properties:** 
      1. **declarations[Component, Directive, Pipe]**
      2. **bootstrap[component list]:** Which components should be recognized in **index.html**
   2. **When importing** component, directive, pipe, no need to add .ts extension as **webpeck** would add.  
      

6. Using Custom Components

1. Nothing in here.

7. Creating Components with the CLI & Nesting Components

1. Command to generate component: **ng generate/g component/c componentName**
   1. This command creates a folder with comopnentName having 4 files. Say component name is **servers**
      1. servers.component.css, servers.component.html, servers.component.spec.ts, servers.components.ts
   2. This Command also registers the component in app.module.ts file in declarations:[] of @ngModule

8. Working with Component Templates

1. @Component🡺template:’Inline\_Template\_Here’ templateUrl:’Relative\_Path\_To\_Html’

9. Working with Component Styles

1. @Component🡺styles:string[ ](Inline CSS Style), styleUrls:string[](Relative Paths)

10. Fully Understanding the Component Selector

1. selector:’app-root’🡸 <app-root>, ’.app-root’ <div class=”app-root” and ’[app-root]’🡺 <div app-root>. Think of string value of selector as css selector.
2. Id and pseudo-selector (hover over) are not allowed in selector property’s value.

13. What is Databinding

1. **DataBinding**: Communication b/w typescript (Component) & its template
2. **3 kinds of DataBinding**:
   1. **From TypeScript to Template**: **String Interpolation** ({{expression}}) **Property Binding** ([propertyName]=”expression”)
   2. **From Template to TypeScript:** Event: (eventName)=”expression”
   3. **Two-Way Binding**:[(**ngModel**)]=”data”

14. String Interpolation

1. {{typeScriptExpression}}🡺{{‘String’}}, {{componentPropertyName}}, {{componentMethodCall()}}

15. Property Binding

1. [domObjectPropertyName]=”expression”;

16. Property Binding vs String Interpolation

1. <p>{{propertyName}}</p> vs. <p [innerHTML]=”propertyName”></p> [DomPropertyName]=”{{do not do this.}}” Mixing Property Binding and String interpolation.

17. Event Binding

1. (eventName)=”expression”. Event Name: Any supported by tag as attribute such as onclick but we will skip the on from onclick.

18. Bindable Properties and Events

1. Do console.log(jsElement) to see which properties and events can be bound. You can bind almost all properties and events.

19. Passing and Using Data with Event Binding

1. <input (input)=”onTyping($event)”> onTyping(event: Event){ (<HTMLInputElement>event.target) } as event.target can contains ref to any DOM object.

20. Important FormsModule is Required for Two-Way-Binding!.html

1. import { FormsModule } from '@angular/forms'; to enable ngModel directive. Imports:[FormsModule] in app.module.ts

21. Two-Way-Databinding

1. <input [(ngModel)]=”serverName”

22. Combining all Forms of Databinding  
23. Practicing Databinding  
24. [OPTIONAL] Assignment Solution

Nothing here

25. Understanding Directives

1. Directives are instructions in the DOM. **Directive = Component – Tempalte**. Typically attribute selector. **@Directive**

26. Using ngIf to Output Data Conditionally

1. All built-in directives use attribute selectors. \* before \*ngIf=”booleanExpression” is for structural directive.

27. Enhancing ngIf with an Else Condition

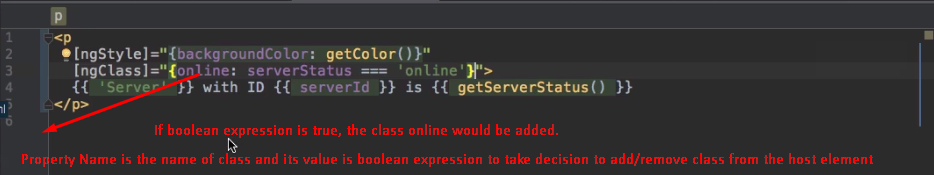
1. <ng-template>: This is built-in Component (directive) shipping with Angular package which can be used to mark places in DOM  
   <p \*ngIf=”booleanExpression; else templateVariable”></p>  
   <ng-template #templateVariable></ng-template>

28. Styling Elements Dynamically with ngStyle

1. **Attribute directive**.  
   

**Above** [ngStyle]

29. Applying CSS Classes Dynamically with ngClass

1.   
   Above [ngClass]

30. Outputting Lists with ngFor

1. <app-server \*ngFor=”let server of servers”></app-server>

31. Practicing Directives  
32. [OPTIONAL] Assignment Solution

Nothing Here.

33. Getting the Index when using ngFor

