Angular 2:

**Components:** A component in angular is a class with a template and a decorator.

**Template**—It is a user interface which contains the HTML, directives and data bindings.

**Class** – it contains required code for the view template. A class in the Angular can contain properties and methods. Properties contain data for the view template and methods contain logic for the view. We use typescript to implement a class in the Angular.

**Decorator:**  it adds meta data to the class making it an Angular component.

Example: example for the class component.

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

template: '<h1>Hello {{name}}</h1>'

})

export class AppComponent {

name: string = "Angular";

}

**Angular Template:**

Inline template: typescript code is not easier to read and understand when it is mixed with the inline template HTML.

The text between the braces is often the name of a component property. Angular replaces that name with the string value of the corresponding component property.

Ex: ‘<h1> {{name}} </h1>’

The text between the braces is a template expression that Angular first evaluates and then converts to string. Angular evaluates all expressions in double curly braces and converts the output to strings, links them with the neighboring strings.

**Template Expressions:**

A template **expression** produces a value. Angular executes the expression and assigns it to a property of a binding which might be an HTML element, component, and directive.

**Angular 2 Nested Components:**

Ex:

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

template: '<div><h1>{{PageHeader}}</h1></div>'

})

export class AppComponent {

pageHeader: string = "Employee Details";

}

In above, since the View Template HTML is just 3 lines we have used an inline template instead of an external template. And also, Angular 2 recommends to extract templates into a separate file, if the view template is longer than 3 lines.

**App Module:** AppModule is the root module which bootstraps and launches the angular application.

**Styling Angular 2 Components:**

Ways for styling angular components

* Styles in external stylesheet: styles.css
* Styles inline in the component HTML file
* Styles in the component html file using <style> tag.
* Specify the styles using the @component decorator styles property
* Specify the styles using the @component decorator styleUrls property.

**Angular Interpolation:**

Interpolation is about the data binding.

The data binding is divided into 3 categories.

* One way data-binding: From Component to View Template
* One way data-binding: From View Template to Component
* Two way data-binding: From Component to View Template & from View Template to Component.

Data-binding is all about moving class property values to a corresponding view template and back.

**Property Binding:**

Both interpolation and property binding flows a data value in one direction i.e. from a class component property to an HTML element property.

**Diff. b/w interpolation and property binding:**

Interpolation is a special syntax that angular converts into a property binding.

To concatenate strings we must use interpolation instead of property binding.

Ex: <img src=’ [http://www.pragimtech.com/{{imagePath}}](http://www.pragimtech.com/%7b%7bimagePath%7d%7d)’ />

To set an element property to a non-string data value, we must use property binding.

**Property Binding:**

Remember to enclose the property name with a pair of square brackets.

Canonical form <button bind –disabled=’isDisabled’>Click me</button>

Angular data binding sanitizes malicious content before displaying it

These bindings flow data in one direction i.e from a component class property to an HTML element property

* Interpolation
* Property Binding
* Attribute Binding
* Class Binding
* Style Binding

Event Binding flows data in the opposite direction i.e from an HTML element to a component.

Syntax for event binding: <button (click)=”onClick()”>Click me</button>

**Pipes in Angular:**

Getting the data could be as simple as creating a local variable or as complex as streaming data over the web socket.

Transform data before display

Built in pipes include lowercase, uppercase, decimal, date, percent, currency etc

To apply a pipe on a bound property use the pipe character “|”.

Ex: <td>{{employee.code | uppercase}}</td>

We can also use chain pipes

Ex: <td>{{employee.dateOfBirth | date | uppercase}}</td>

Pass parameters to pipe using colon “:”

Ex: <td>{{employee.dateOfBirth | date: ‘fullDate’}}</td>

We can also create custom pipes.

Custom Pipes in Angular: we can create our own custom pipes.

**Steps to create the custom pipe:**

Create the custom pipe. (employeeTitle.pipe.ts)

Declare the custom pipe in the module where we need it. (app.module.ts)

Use the custom pipe just like any other built-in angular pipe. (employeeList.component.html)

**Input and output Decorator:**

**@Input:** @Input is used to define an input property to achieve component property binding.

This decorator binds a property within one component to receive a value from another component ( Parent->Child)

It can be annotated at any type of property like number, string, array or user defined class.

**@Output:** @Output is used to define an output property to achieve custom event binding.

This decorator binds a property of a component to send data from one component to calling component ( Child -> parent)

It can also alias the property name as @Output(alias), this alias name will be used in custom event binding in calling component.

**Custom Directives in Angular 2:**

Components are high-order directives with templates and serve as building blocks of angular applications.

Directives are 3 types.

* Components – directives with templates
* Structural Directives – directives that creates and destroys DOM elements
* Attribute Directives – directives that manipulate DOM by changing behavior and appearance

**Attribute Directives:**

In this, as the name goes, are applied as attributes to elements. Thease are used to manipulate the DOM in different ways except creating or destroying them.

By using these directives, we can achieve one of the following task.

* Apply conditional styles and classes to elements
* Hide and show elements, conditionally.
* Dynamically changing the behavior of a component based on a changing property.

**Structural Directives:**

These are not DOM-friendly it means they can create, destroy or re-create DOM elements based on certain conditions.

This is the main diff. b/w hidden attribute directive does. This is because hidden retains the DOM element but hides it from the user, whereas structural directives like \*ngIf destroy the elements.

Directives are other Angular 2 members created as a class. This classs is then decorated with the Directive decorator which is imported from the @angular/core.

**@Injectable and @Injector:**

**@Inject:** @Inject () is a manual mechanism for Angular 2 that a parameter must be injected.

When using typescript, @Inject is only needed for injecting primitives.

**@Injectable ():** It is a class that can be used with the dependency injector. And also it is not strictly required if the class has other Angular 2 decorators on it or doesn’t have any dependencies.

**Providers:** A provider is a resource or javascript thing that angular uses to provide something we want to use to generate the result.

* A class provider provides an instance of the class
* A factory provider provides whatever returns when you run a specified function.
* A value provider doesn’t need to take an action to provide the result like the previous two, it just returns its value.

We can add a provider to a component by creating a class provider using MyClass, that will generate the instance we want to use in our application.

**Promises:** Promises make it easier to write asynchronous code compared to using callbacks, and many libraries and web APIs return promises for asynchronous operations like fetch().

**Differences b/w Promises and Observables:**

**Promises:**

* Emits a single value.
* Not Lazy.
* Cannot be cancelled

**Observable:**

* Emits multiple values over a period of time.
* Lazy. An observable is not called until we subscribe to the Observable.
* Can be cancelled by using the unsubscribe() method.
* Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc.