**Q: What are components? OR  
Q: What are the Contents of component?**  
Ans: Components are the most basic building block of an UI in an Angular application. An Angular application is a tree of Angular components. Angular components are a subset of directives.

A component must belong to an NgModule in order for it to be usable by another component or application. To specify that a component is a member of an NgModule, you should list it in the declarations field of that NgModule.

Component is made up of 3 parts, as follows:

1. Template: Template represents the view for component. It creates using HTML & will be the user interface for your application.
2. Metadata: every component has some metadata attached to it in the form of decorator. It provides more information about the component. The metadata is a feature of typescript. A decorator is just a function that provides information about a class attached to it. Component decorator contains useful **configuration** properties like selector that is used to define custom tags through which we can render our component in browser, templateUrl or template, styleUrls or styles etc.
3. Class: Class is nothing but code that supports the view. This is created using typescript. The class contains properties and methods that can be used to control the logic of the view. For example we can have method to show or hide an element based on value of a property.We need to export our class in order to reuse it in other components**.**

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Q: How can we manually create component?**

**A:**

1. **In angular 5:**

Normally we use **ng generate component** command to create a component. If you want to create a component from scratch without CLI , The steps would be:

1. Go to app folder in src, create a new folder called ‘***components’***. Within that folder we will generate our new component. We can create a component without folder too, but it is always good practice to put all your components in a particular folder. Within that folder we will create folder for every component, let’s say we have to create **user** component then our folder name will be user.
2. Then create component.ts file, in this case user.component.ts file
3. import Component from angular/core module.  
   syntax would be: **import { Component } from '@angular/core';**
4. Then we need to create decorator. It is typescript feature.it basically adds extra information about our component. Often it contains selector property, template/templateUrl property and styleUrls/styles property. These are kind of basic properties. We can add more properties if it is required. Selector property contains the value that can be used as custom html tag to render our component in browser. Selector value needs to attach prefix **app-**. Since our component Name is ***user***, our selector value will be **app-user**.
5. Then define a class. In this case, className would be UserComponent. Classname should be in pascal case. Pascal case starts with an uppercase and every word after that will start with uppercase letter. and Since our component needs to be accessed from outside, we need to use ***export*** keyword along with classname.
6. Whenever we create any new component, we need to import it & register it into ***appModule.ts*** file. And then into declarations array of @ngModule, add our custom component as an array element.

**Syntax: import { UserComponent } from ‘filePath’;**

1. Then into @ngModule section’s declartions array, add our new component name.
2. Now go to appComponent.html file, and embed your component using value of selector property specified in component decorator. It will render your new component in browser. That’s it.
3. **In angular 2:**

First 6 steps are same as above, seventh step would be:

7) Since in angular2, there is lack of appComponent.html, hence to display the custom component in browser, we insert it into template metadata/property of decorator in same appComponent.ts file .

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------  
  
**Q: What are Services and how are they useful?**

Ans: Service is class with a specific purpose. The service has basically one job, to provide data to the components that asks for it. Let’s say we have two components displayEmployeeName and displayEmployeeDetails. Both components are displaying the data from EmployeeData Array. Since one component cannot access data from another component, we will have to define employeeData in both the components. Means we are defining same array in 2 components. It violates the Single Responsibility Principle. And also it violates Single Responsibility Principle because components responsibility is define view logic and do not store data that component displayed in the browser.

**Steps of creating and implementing service in application:**  
1) create a ts file in which we are going to define our service logic.

2) define a class and within that class create a method that return an array which hold the employee data. Use export alongwith className so we can access the class outside the file

3) Then insert @Injectable decorator to decorate your service and import injectable at the top  
import { Injectable } from ‘@angular/core’;

4) Then register your service with angular injector. Go to appComponent.ts file, and within providers metadata of decorator, insert your serviceClass as an array element. And import the service at the top.Through this step we are telling angular injector, provide this service to anyone who ask for it.

5) Then inside class of both components define a property that hold empty array. We will use this property to render the data

6) Now final step is declare the service as dependency so Go to component.ts file, Dependency goes into constructor. Define constructor function, if it is unavailable. And within parenthesis of constructor function, create local instance of your service. For ex:  
constructor(private \_employeeService: EmployeeService) {};

EmployeeService is our service class and \_employeeService is our local instance. Private is our access modifier.  
\*\*Then import the service at the top.

7) Then to fetch data from the service, we need to write a code into ngOnInit lifecycle hook method. Firstly we need to implement our ngOnInit method by our class. For ex:  
export class Employee implements OnInit { class code }.

Then into ngOnInit method, assign serviceMethod to our property that holds empty array to fetch the data that method returns.

For ex:  
this.employees = this.\_employeeService.getEmployeeData();  
employees is our class property that initialized as an empty array earlier, \_employeeService is service instance through which we can access service and getEmployeeData is service method which returns the required employee data.

**Advantages of services are as follows:**

1. Share the same data across multiple components: It we have one array that is required in 2 components. In that case instead of define that array in both components, create a service to define that array. And then through DI/dependency injection we can use service.
2. Implement application logic:
3. External interaction: connecting to database.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Q: What is Dependency Injection?**

**Dependency Injection (DI)** is a way to create objects that depend upon other objects. A Dependency Injection system supplies the dependent objects (called the dependencies) when it creates an instance of an object.

**Q: What is component interaction?  
Ans:** Component interaction means passing the data from Parent Component to Child Component or vice-versa. To achieve such interaction we use inputs and outputs.  
  
To accept the data from parent; we use inputs decorator in Child component. And to send the data to Parent Component; we use outputs decorator in child component.

**A) Passing data from Parent Component to Child Component:**

i) Define property in parent Component which we are going to pass,  
 for ex: public parentProp: string = ‘Hi all;

ii) Then in child Component, specify inputs as metadata in decorator.

**@component({  
 inputs: [‘parentData’];  
})**

iii) Now define a property inside child component class to hold the value passed by parent component. Property name should be similar as the name specified in inputs decorator. In this case property name should be parentData;

public **parentData**: string;

iv) Now go to child component custom tag defined in parent component, and inside start tag, assign parent component property value to child component property using binding as follows:

<child [parentData]=’ parentProp’></child>  
  
Now you will see the value of parentProp has been assigned to parentData child component property.

**A) Passing data from Child Component to Parent Component:**

1. Since we are passing data from child component, we need to insert outputs metadata property inside child component. Ex:

**@Component ({**

**Outputs: [ ‘childEvent’ ]**

**})**

1. In child component class, create instance of Eventemitter. For ex: childEvent = new EventEmitter<string>();
2. Then import EventEmitter at the top of child component  
   For ex: import { EventEmitter } from ‘@angular/core’;
3. Now to pass the value to parent; we have to trigger an event in child through this step, we will call one particular child component method and then within that method we will call emit method along with our EventEmitter instance to pass the data.  
   Suppose we have textfield in our child component and we are passing its entered value to parent component, we will have to create reference variable for texfield first and then on keyup or keydown event we will pass the value to parent.

For ex: <input type=”text” #childtext (keyup)=’onChange(childText.value)’>  
in above code, childText is reference variable and on keyup event we are calling onChange method

1. Now in onChange method code, we will pass entered value to parent using emit method along with EventEmitter instance  
   For Ex: this.childEvent.emit(value);
2. Now in **parent component class**, create a property that is going to hold data passed by child component. For ex: public childData : string;
3. Then where we are calling our child component in Parent component, trigger EventEmitter instance(childEvent), just assign value of $event to public component property which is created to receive the data from child.

For ex: <child (childEvent)=’childData=$event’></child>   
$event value will hold the value that is passed by child component.

Now we can use child component property into parent component.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

What is interface? How to implement it?

One of TypeScript’s core principles is that type-checking focuses on the *shape* that values have. This is sometimes called “duck typing” or “structural subtyping”.

*In TypeScript, interfaces are the most flexible way of describing types, create shorthand names for commonly-used types, constrain class implementations, describe array types, and more.****While they don’t generate any code (and thus have no runtime cost!)****, they are often the key point of contact between any two pieces of TypeScript code, especially when working with existing JavaScript code or built-in JavaScript objects.*

**Interface’s job is define “shape” of objects,**helps us keep our programs error-free by providing information about the shape of the data we work with. Interface is powered by some concepts to teach type checker how to verify object with the expected properties is being used:

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#### Q: How can we create angular app? ng new applicationName

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Q: How many types of directives available in angular?**  
There are three kinds of directives in Angular:

1. Components—directives with a template.
2. Structural directives—change the DOM layout by adding and removing DOM elements.
3. Attribute directives—change the appearance or behavior of an element, component, or another directive.

**------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Q: Explain Structural Directives?  
  
Ans: Structural Directives are the directives that let you add or remove html elements from DOM.**

**3 Common structural directives are:  
 a)ngIf**

**b)ngSwitch**

**c)ngFor**

**ngIf and ngSwitch directives are used to conditionally render html elements whereas ngFor is used to render a list of html elements like li element in html.  
  
ngIf examples:**<div \*ngIf = "visible">shown</div>

<div \*ngIf = "!visible">Hidden</div>

<br/>

**ngIf with else statement [ using ng-template element for else block ]:**

<div \*ngIf = "visible; else elseBlock">shown</div>

<ng-template #elseBlock>Else Block</ng-template>

<br/> **ngIf with then and else block:**

**[ using ng-template element for then and else block ]**

<div \*ngIf = "visible; then thenBlock; else elseBlock2"></div>

<ng-template #thenBlock>Then Block</ng-template>

<ng-template #elseBlock2>Else2 Block</ng-template>

**ngSwitch:**  
**If there are multiple conditions and out of those conditions, one condition is true and we want to perform operation in order to that condition, in that case ng-switch is good option to use.**  
For Ex:  
angular5: into html  
<div [ngSwitch]="color">

<div \*ngSwitchCase="'red'">You picked red color</div>

<div \*ngSwitchCase="'green'">You picked blue color</div>

<div \*ngSwitchCase="'blue'">You picked green color</div>  
 <div \*ngSwitchDefault>Please pick either red, green or blue</div>

</div>  
  
**\*\*Note: use \*ngSwitchWhen instead of \*ngSwitchCase in angular2.**  
  
into ts file, in class  
**color: string = 'red';**  
   
Output: **You picked red color**

**Note:**

1. **False cases will not be generated in DOM.**
2. **\*ngSwitchCase value will need to write in two quotes: \*ngSwitchCase="'red'"**
3. **ngSwitch does not have asterisk included.**

\*ngFor:  
\*ngFor is used to render a list of elements. We use \*ngFor for retrieve. We often use \*ngFor to fetch data from an array.  
  
Suppose we have an colors array that contains 4 colors  
colors = [‘red’, ‘green’, ‘blue’, ‘black’]  
If we need to display those array elements into our application, We will have to use \*ngFor. our \*ngFor syntax would be as follows:  
**Syntax:**

\*ngFor = ‘let color of colors’  
here let is ES6 feature that indicates block level scope, color refers to each element in the array during the iteration, and colors is our array.  
  
\*ngFor repeats your element as per your array length. For Ex. if in html we have attached \*ngFor to li, that li element will be generated as per the length of an array. In above case 4 li elements will be generated.  
  
We can access index numbers of elements too:   
\*ngFor = ‘let color of colors; index as i’;  
i is a variable that we use to display index numbers here {{ i }}..index starts with zero so first element will be display its index as zero.  
  
\*\*Note: angular2 does not support displaying index using index as i syntax. But It works in angular5.

We can check whether the element is first or not:  
\*ngFor = ‘let color of colors; first as f’;  
f is a variable that we use to check whether element is first in the array or not.

{{ f }}. It will return boolean value, either true or false.  
  
  
We can check whether the element is last or not:  
\*ngFor = ‘let color of colors; last as l’;  
l is a variable that we use to check whether element is last in the array or not.  
{{ l }}. It will return boolean value, either true or false.  
  
We can check whether the element is even or odd:  
**<div \*ngFor = "let color of colors; odd as o">{{ o }}: {{ color }}</div>  
<div \*ngFor = "let color of colors; even as e">{{ e }}: {{ color }}</div>**

**First statement will return true if element has odd index otherwise it will return false,   
and Second statement will return true if element has even index otherwise it will return false.**

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Q: What are attribute directives?**

Attribute directives are used as attributes of elements. [NgStyle](https://angular.io/guide/template-syntax#ngStyle) and ngClass directives are example of attribute directives.  
  
  
ngClass example: It adds CSS classes based on the Boolean value of property. If property is true then that class will be attached otherwise class will not be attached.  
Ex:   
<p [ngClass]="{classOne: cOne, classTwo: cTwo}">ngClass Exmaple:</p>  
  
**within Class:**  
 public cOne = true;

public cTwo = true;  
  
Output: Since both properties are true; Both classOne & classTwo classes will be attached to p element. If we assign false value to both properties, those classes will not be attached.  
  
  
  
**ngStyle example:** It adds CSS styles based on the property.

<p [ngStyle]="{'font-size':textSize, 'color':textColor}">ngStyle Exmaple:</p>  
  
**Define below properties in class:**   
public textSize = '30px'; public textColor = 'darkblue';

**Output: Above p will have 30px font-size and text-color will be darkblue.**

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------  
  
**Q: How angular compiles our application?**  
Ans: just type npm start or ng serve command  
  
**Q: What is typescript?**  
ES6 is the current version of JavaScript. TypeScript is a superset of ES6, which means all ES6 features are part of TypeScript, but not all TypeScript features are part of ES6. Consequently, TypeScript must be transpiled into ES5 to run in most browsers.

One of TypeScript's primary features is the addition of type information, hence the name. This type information can help make JavaScript programs more predictable and easier to reason about.

Types let developers write more explicit "contracts". In other words, things like function signatures are more explicit.

Without TS:

function add(a, b) {

return a + b;

}

add(1, 3); // 4

add(1, '3'); // '13'

**With TS:**

function add(a: number, b: number) {

return a + b;

}

add(1, 3); // 4

// compiler error before JS is even produced

add(1, '3'); // '13'

**Q: Typescript Features**

Ans:

* **Types**
* **Interfaces**
* **Shapes**
* **Decorators**

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------  
  
**Q: How can we define array in typescript?**

## \*Declaring and Initializing Arrays

To declare an initialize an array in Typescript, use the following syntax −

### Syntax

var array\_name[:datatype]; //declaration

array\_name = [val1, val2, valn..] //initialization

For Ex  
**var myArr : number[] // declare number array  
var myArr : string[] // declare string array  
var myArr : any[] // declare any type array**  
  
An array declaration without the data type is deemed to be of the type any. The type of such an array is inferred from the data type of the array’s first element during initialization.  
  
Arrays may be declared and initialized in a single statement. The syntax for the same is −

var array\_name[:data type] = [val1,val2…valn]

For Ex:  **var numlist:number[] = [2,4,6,8]**

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------  
  
**Q: How can we manipulate DOM elements in angular?**  
Ans: DOM manipulation contain several useful things like changing the appearance of DOM elements, add or remove classes/Ids or attributes from html elements, Change style of an element and so on. To achieve DOM manipulation we need to access our html elements.  
  
We use ViewChild or ViewChildren decorator to access the template through which we can perform DOM manipulation. To get the reference of html element we define template reference variable in the required element and then by using ViewChild/ViewChildren decorator we can access that component into our class.  
  
Steps to manipulate the DOM using ViewChild:

1) First add template reference variable to your html element, suppose we are accessing DIV element, then add template reference variable to it.  
For ex. <div #refVariable>We are inside template..</div>

2) Now go to component.ts file and import ViewChild from angular/core

For ex. import { Component, OnInit, ViewChild } from '@angular/core';

3) Now within component class, define your local property and attach it ViewChild decorator along with element reference variable as below:  
 @ViewChild('refVariable') refVariable;

first 'refVariable' is template reference variable whereas second refVariable is class property. Class property holds reference of required DOM element. You can name anything you want to your property.

4) Now in **ngOnInit** method, we can access our required html element. To access the element use below syntax:  
Syntax: this. refVariable.nativeElement;  
refVariable above is our class property that hold reference of our html element.

Now we can easily Modify our html element as follows: **1) Changing CSS style of an element:**

this.refVariable.nativeElement.style.background = "yellow";  
 this.refVariable.nativeElement.style.display = "none";

**2) adding class class1 to element:**

this.refVariable.nativeElement.className = "class1"  
this.refVariable.nativeElement.className += " class2"

**3) assigning ID to the element:**

this.refVariable.nativeElement.setAttribute("id", 'myId');  
  
  
NOTE: 1) Remember we cannot access DOM elements into constructor.

2) We can use ngAfterViewInit decorator too, to access the template elements.

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Q: component lifecycle hooks of components?  
  
Ans: A component has a lifecycle managed by angular. Angular;**

* **Creates the component**
* **Renders the component**
* **Creates and renders the component children**
* **Checks when the component data-bound properties changes and**
* **Destroys the component before removing it from the DOM  
    
  There are several lifecycle hooks; ngOnChanges, ngOnInit, ngDoCheck, ngAfterContentInit, ngAfterContentChecked, ngOnDestroy etc.**

**The 3 most commonly used hooks are**

|  |  |
| --- | --- |
| **Life Cycle Hook** | **Purpose** |
| **ngOnChanges** | ngOnChanges executes, every time the value of an input field changes.This hook contains SimpleChanges object that hold current and previous values of input field. This gets called before ngOnInit method |
| **ngOnInit** | ngOnInit executes after the constructor and after ngOnChanges hook. It is most commonly used for component initialisation and retrieving data from a database |
| **ngOnDestroy** | Executes just before angular destroys the component and generally used for performing cleanup. |

**------------------------------------------------------------------------------------------------------------------------------------------------**

**Q: What is routing? Routing steps?**

**Ans:** The Angular [Router](https://angular.io/api/router/Router) enables navigation from one [view](https://angular.io/guide/glossary#view) to the next as users perform application tasks.

**Q: What is lazy loading?**

**Ans:** Lazy loading modules help us decrease the startup time. Lazy loading simply means the ability to load different pieces on demand. With lazy loading our application does not need to load everything at once, it only needs to load what the user expects to see when the app first loads. Modules that are lazily loaded will only be loaded when the user navigates to their routes.

# Q: 2 types of compilation? Ans:

# 

## Angular compilation

Angular offers two ways to compile your application:

1. Just-in-Time**(JIT)**, which compiles your app in the browser at runtime
2. Ahead-of-Time**(AOT)**, which compiles your app at build time. The Angular Ahead-of-Time (AOT) compiler converts your Angular HTML and TypeScript code into efficient JavaScript code during the build phase before the browser downloads and runs that code.

JIT compilation is the default when you run the build-only or the build-and-serve-locally CLI commands:

ng serve

For AOT compilation, append the --aot flags to the build-only or the build-and-serve-locally CLI commands:

ng serve --aot

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Que: Explain custom directives?**

**Ans:** Custom directives mean creating our own directives other than inbuilt angular directives. **Custom directives steps would be:**

1. **Create custom directive through CLI:  
   ng generate directive directiveName  
   Note:** if you create custom directive using CLI, it will update **appModule.ts file** and it will create ts file containing class with directive decorator and constructor. You will only need to put your code inside class.And import ElementRef at the top**.**

**B) Steps to manually create custom directives**

1) Create one ts file that import Directive and ElementRef from angular/core and add Directive decorator and also create the class for the Directive: We use ElementRef to control our DOM element.  
Lets say we are creating custom directive that has background-color yellow:  
  
Code:

import {Directive, ElementRef} from '@angular/core';

@Directive({

selector: '[yellowBg-directive]' // directive as an attribute

})

export class yellowBgDirective {

constructor(el: ElementRef) { // el is reference for ElementRef

el.nativeElement.style.background = 'yellow';

}

}

2) Then import custom directive within appModule.ts file and add into declarations array of ngModule decorator like we do for component.

3) Now add custom directive selector into appComponent.html or any necessary template to display within browser.

<h1 yellowBg-directive>My Custom Directive</h1>

OR

<h1 class='yellowBg-directive'>My Custom Directive</h1>

OR

<yellowBg-directive>My Custom Directive</yellowBg-directive>

Since we have added custom directive to h1 element, yellow background-color will be applied to h1. We can invoke our custom directive as element or attribute or class.

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Q: Explain @HostListener?**

In Angular, the @HostListener() function decorator allows you to handle events of the host element in the directive class.

We can use @HostListener() decorator with directives to trigger the events and perform some kind of operations. Remember if we trigger event inside parent component using **HostListener**, same event will get propagated inside its child component too.   
   
**\*\*How to implement @HostListener**  
  
1) first import hostlistener from angular/core inside ts file of directive:  
**import { Component, HostListener } from '@angular/core';**

2) Include HostListener along with the event and the function name; It simply means on triggering that event we are calling the function next to it.  
 **@HostListener('click') doSomething() {**

**alert('Hi All..');**

**}**

**--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Q: Difference between component and directives?**

The basic difference between a component and a directive is that a component has a template, whereas an attribute directive, or structural directive or custom directive does not have a template.

**Q: Explain http and observables?**

**Q: Explain architecture of angular2? OR**

**Explain important aspects of angular2?**

There are mainly 5 important aspects of angular. They are as follows:

1) Modules

2) Components

3) Directives

4) Services

5) Routers