1. If in case angular dependencies do not work or npm start gives error then execute below commands from the root folder of your project which had the package.json file
   1. Delete node\_modules folder
   2. npm cache clean – force
   3. npm i --save
2. Folder structure:
   1. Package.json => dependencies , =scripts to run
   2. Angular.json => all the information for the angular kick start => initialization process
   3. Index.html => inside the src folder => <app-root>
   4. Styles.css => global styles
   5. Main.ts => reference the bootsrap module
   6. App.module.ts => decorated with @NgModule
      1. Declarations:[] => user defined components /pipes/ directives
      2. Imports:[] => all the prededined modules of angular
      3. Providers: [] => services
      4. Bootstrap => the name of the root component
      5. AppComponent
         1. Html => UI for the component
         2. Css => styles specific to this component
         3. Ts => the BL to modify the DOM
         4. Spec.ts => test cases
3. Install angular cli  
   npm i @angular/cli@<version> => install locally within your workspace  
   npm i -g @angular/cli@latest => install globally on your system
4. Create new angular project  
   ng new <projectname>
5. To create a component  
   ng g c <componentname>
6. 1-way Data Binding syntax => from component to view
   1. {{}} – string interpolation [ always evaluates values in string ]
   2. [] -> it retains the type of the variable
7. Event binding : (click)
8. 2-way data binding : banana in a box [()]
   1. Use ngModel directive provided by FormsModule  
      **DO NOT FORGET TO ADD FormsModule IN YOUR app.module.ts file**
9. Style binding  
   [style.<style\_property>] = “’<value>’”

[style.<style\_property>] = “<propertyname>”

1. ViewEncapsulation => provides how styles will flow from parent to child
2. Nested Components => place the selector in the components the hierarchy of the DOM is created  
   APP => <header> <main> <footer>  
   Main=> <product-list>  
   ProductList => <product>
3. Parent - child communication
   1. Data can be send from parent to child using : @Input() decorator
   2. Child sends event to parent component using : @Output()
4. Directives
   1. Components => UI (HTML) + BL
   2. Attribute directive => ngModel, style, ngStyle , class, ngClass,   
      modify the behaviour of the html element on which it is applied
   3. Structural => \*ngIf, \*ngFor, ngSwitch=> \*ngSwitchCase and \*ngDefault
   4. Custom directive :
      1. Create a directive  
         ng g d hover  
         ng g directive hover
      2. Selector has syntax of [appHover]
      3. To access the element on which the directive is used :  
         ElementRef
      4. To change any style :  
         Renderer2
      5. To change the property of host element:  
         @HostBinding
      6. To list for any event change on host element :  
         @HostListener
      7. Directives can take input from host element using @Input()
      8. To use directive without data binding  
         <img appHover/>
      9. To use directive for @Input(), use data binding syntax to pass the data  
         <img appHover [itemname]=”itemname”/>
5. Lifecylce methods:
   1. Constructor => to instantiate the component. DO NOT DO ANY LONG RUNNING PROCESS IN THE CONSTRUCTOR
   2. ngOnInit => gets called only once and used for initialization
   3. ngOnChanges => listens for changes in properties decorated with @Input(). It listens only for changes in primitives
   4. ngdoCheck => listens for changes in properties decorated with @Input(). It listens only for changes in primitives or objects
   5. ngOnDestroy => gets called once when the component is removed from the DOM
6. Pipes => used to format the data while displaying on the html template. THEY DO NOT MODIFY THE ORIGINAL DATA
   1. Built-in:
      1. Uppercase
      2. Lowercase
      3. Json
      4. Number
      5. Currency
      6. Date
      7. Slice
   2. Custom pipe => @Pipe [ transform(value, …arguments)
7. Template driven forms
   1. Add FormsModule in app.module.ts
   2. ngForm => directive exposed on the the <form #empform=”ngForm”>
   3. databinding => ngModel
   4. using ngModel can have access to various properties of form template for validations : touched, dirty, invalid etc…
8. Angular animations
   1. Execute below command from the root package  
      npm i @angular/animations
   2. RESTART THE PROJECT
   3. Add BrowsersAnimationModule in app.module.ts within imports[]
   4. Use animations[] within the component to apply styles and transitions
9. Reactive Forms
   1. Import ReactiveFormsModule in app.module.ts
   2. Create FormGroup object
   3. Create FormControl objects that are a part of the form group
   4. Custom Validators => create a custom function
10. Consuming REST API
    1. HttpClientModule import in app.module.ts
    2. Create a service and inject HttpClient in the constructor of the service
    3. HttpClient has various methods => get, post, put, delete
    4. These methods returns a reference of type Observable
    5. Subscribe to the observables to fetch data over REST API
11. Angular Material
    1. npm install --save @angular/material@12.2.13 @angular/cdk@12.2.13
    2. Add below in styles.css  
       @import '~@angular/material/prebuilt-themes/indigo-pink.css';
    3. Add below in index.html  
       <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    4. Add below in app.module.ts  
       import {MatButtonModule} from '@angular/material/button';

import {MatCardModule} from '@angular/material/card';

import {MatInputModule} from '@angular/material/input';  
  
DO ADD THE RESPECTIVE MODULES IN imports[]

1. Implement routing in angular
   1. Routing yes when you create project for the first time or use ng new <projectname> --routing
   2. Index.html => <base href=”/”>
   3. App-routing.module.ts
      1. Import Routes and RouterModule
      2. Routes[] => provide the paths and respective components for the path
      3. Load the paths using RouterModule.forRoot()
      4. Add <router-outlet> tag where the routed component needs to be displayed
      5. routerLink to configure link for a path added in <a> tag => static routing
      6. programmatic or dynamic routing :
         1. import Router class in the compoinent that need to redirect to a specific path programmatically and use navigate() passing array of url
      7. Path parameters => required parameters  
         <http://localhost:4200/employees/1>  
         in app.module.ts  
         path:’employees/:id’
      8. To access path paramters
         1. Inject ActivatedRoute in the constructor
         2. Use params.map() function to fetch the id passed
      9. Query parameters or optional parameters [ NO NEED TO CONFIGURE IN THE app.module.ts file since its optional ]

[http://localhost:4200/employees;id=1  
Inject Router  
router.navigate([‘’, {id:<val>}])](http://localhost:4200/employees;id=1Inject%20Routerrouter.navigate(%5b‘’,%20%7bid:%3cval%3e%7d%5d))http://localhost:4200/employees?id=1  
Inject Router  
router.navigate([‘’,], {queryParams: {id:<val>}})

* + 1. Relative url  
       router.navigate([‘’,], {relativeTo:this.route,queryParams: {id:<val>}})
    2. Nested Routes  
       {path:’employees/:id’,component:ProfileComponent,  
       children:[{path:’info’, component:ProfileinfoComponent}]}  
       DO ADD <router-outlet> IN THE ProfileComponent FOR CHILDREN TO DISPLAY
    3. Route Guards
       1. CanActivate => to protect the routes
       2. HttpInterceptor => to intercept the request