**Angular cli commands:**

Ng new <project>:

To create new project

## ng generate:

## generate a new component. You can also use ng generate directive|pipe|service|class|guard|interface|enum|module

**ex:**

ng g m timecard/timesmodule –routing

ng g component foo/fooList --module=foo

ng build:

It compiles an Angular app into an output directory named dist/ at the given output path.

## Ng build –prod //build in production enviranment

ng serve:

for build and run the application.

## ng test:

## to execute the unit tests via [Karma](https://karma-runner.github.io/).

## ng e2e:

## It is used to build and serve an Angular app, then runs end-to-end tests using Protractor.

## Ng lint

## ng lint run the linting tool on angular app code. It checks the code quality of angular project specified. It uses TSLint as default linting tool and uses the default configuration available in tslint.json file

## Ng – version

## Get version of angular application

NPM commands:

node –v

to get version of node  
npm –v

to get version of npm

## npm init:

## for create package.json file

## npm install:

## it install packages from package.json in  node\_modules is the folder

To install specific package :

npm install <package\_name>@version

Update:

npm update <package\_name> for specifc package

npm update //for all packages

uninstall package:

npm uninstall <package\_name> -g

npm uninstall <package\_name>

To get the list of installed packages, use the command

npm list

help:

npm help

You can also install packages as a developer dependency i.e., these packages are only needed for development

npm install <package\_name> --save-dev

package.json -Contains the dependencies of the angular application

index.html -This is the first page that gets loaded when we run the application

node\_modules -All the node module dependencies are created in this folder

angular.json -configuration file for angular cli

tsconfig.json -config file for typescript

assets, -this could be anything from images, videos, json files to styles and scripts

*package-lock.json -* This file contains the exact version of the package, unlike *package.json*which contains the semantic version

A globally installed packages works anywhere on the machine. To install global packages you’ve to use -g flag.

**Environments files:**

Those are different **build configurations files** for your project, such as stage and production, with different defaults.

A project's src/environments/ folder contains the base configuration file, environment.ts, which provides a default environment. You can add override defaults for additional environments, such as production and staging, in target-specific configuration files.

**Angular application Flow:**

main.ts  >>   app.Module.ts  >>  app.component.ts  >>  index.html  >>  app.component.html

Step 1: main.ts file gets loaded. It bootstraps(starts) the application by calling App.module.file.

Step 2: app.module.ts file holds an array of bootstrap components. Here, we find our root component reference.

Step 3: Root component gets loaded and the template files from app.component.html become part of index.html.

**Life cycle hooks:**

|  |  |
| --- | --- |
| ngOnChanges() | This event called every time when any data bound property within the component has been changed. The method receives a [SimpleChanges](https://angular.io/api/core/SimpleChanges) object of current and previous property values.  Called before ngOnInit() and whenever one or more data-bound input properties change. |
| ngOnInit() | Initialize the directive/component and sets the directive/component's input properties.  Called once, after the firstngOnChanges(). |
| ngDoCheck() | It is called during every change detection run(one of  Zone hooks get called,), Angular has an internal system that goes around the component processes every so often looking for changes that the compiler cannot detect on its own  Called during every change detection run, immediately after ngOnChanges()and ngOnInit(). |
| [ngAfterContentInit()](https://angular.io/api/router/RouterLinkActive#ngAfterContentInit) | Called after Angular projects external content into the component's view / the view that a directive is in.  Called once after the first ngDoCheck(). |
| ngAfterContentChecked() | Called after Angular checks the content projected into the directive/component.  Called after the [ngAfterContentInit()](https://angular.io/api/router/RouterLinkActive" \l "ngAfterContentInit) and every subsequent ngDoCheck(). |
| [ngAfterViewInit()](https://angular.io/api/forms/NgForm#ngAfterViewInit) | Called after Angular initializes the component's views and child views / the view that a directive is in.  Called once after the first ngAfterContentChecked(). |
| ngAfterViewChecked() | Called after Angular checks the component's views and child views / the view that a directive is in.  Called after the [ngAfterViewInit()](https://angular.io/api/forms/NgForm" \l "ngAfterViewInit)and every subsequent ngAfterContentChecked(). |
| ngOnDestroy() | Cleanup just before Angular destroys the directive/component. Unsubscribe Observables and detach event handlers to avoid memory leaks.  Called just before Angular destroys the directive/component. |

**Constructor:**

A constructor is a special method which will be called whenever we create new objects. And generally used of initializing the class members.

**Some of important modules:**

BrowserModule @angular/platform-browser When you want to run your app in a browser

componentModule

NgModule

Injectable

Directive

Pipe

CommonModule @angular/common When you want to use NgIf, NgFor

FormsModule @angular/forms When you want to build template driven forms (includes NgModel)

ReactiveFormsModule @angular/forms When you want to build reactive forms

RouterModule, Routes @angular/router When you want to use RouterLink, .forRoot(), and .forChild()

HttpClientModule @angular/common/http When you want to talk to a server

needed for reactive forms : import{FormGroup,FormControl,Validators,FormBuilder} from from '@angular/forms';

**Routing:**

Routing helps in directing users to different pages based on the option they choose on the main page. Hence,

based on the option they choose, the required Angular Component will be rendered to the user.

<router-outlet> </router-outlet>.--container for display route views, it is directive from routermodule

forRoot creates a module that contains all the directives, the given routes, and the router service itself.

forChild creates a module that contains all the directives and the given routes, but does not include the

router service. It registers the routers and uses the router service created at the root level.

**Angular version wise Features :**

Angular 2:

Angular 2 is components based

used for develop mobile driven apps

no longer controllers and $scope

Support for TypeScript

multimple browser support

Angular 4:

Faster and smalle

template is now ng-template

httpclient

useses routermodule- forroute, forchild for routing

Angular 5:

Router Life Cycle

compiler improvements

RxJs Support

Angular 6:

Angular Elements

RxJs 6 Support

| **Angular 6** | **Angular 7** | **Angular 8** | **Angular 9** |
| --- | --- | --- | --- |
| Angular Element | CLI Prompts | Ivy Engine | Default Ivy in v9 |
| Service worker | Virtual Scrolling | Web Workers | Phantom Template Variable Menace |
| Internationalization (i18n) | Drag and Drop | Lazy Loading | Dependency Injection Changes in Core |
| Bazel Compiler | Bundle Budget | Improvement in ng-upgrade | Service Worker Updates |
| ng-add / ng-update | Angular Compiler | Support for Node 10 | i18n Improvements |
| ng-update | Angular Do-Bootstrap | CLI workflow improvements | More reliable ng update |
| ngModelChange | Better Error Handling | Upgrading Angular Material | API Extractor Updates |
| TypeScript 2.7 support | TypeScript 3.1 support | TypeScript 3.4 support | Typescript 3.7 support |
| Improved decorator error messages | New ng-compiler | Differential Loading | Component Harness |
| <ng-template> updated to <template> | Native Script | Improved Web Worker Bundling | ModuleWithProviders Support |

**Subjects:**

Subjects are specific kind of observable , they are multicast means we can subscribe subject observables in multiple components and also we can update the subject observable stream with next(),error, complete method

subject are multicast. but observables are singlecast.

we can update the subject observable but regular observable does not.

**Behavior Subjects:**

When we subscribe to a behavior subject, it will give you the last emitted value right away.

Behavior Subject is similar to subject but only difference is that we can set the initial value.

const subject =new BehaviorSubject(0);

In Subject we cannot set an initial value but in Behavior subject we have to set an initial value otherwise it will throw an error.

**Replay Subjects:**

Replay Subjects keep a given number of historical values so that those values can be replayed to new subscribers.

**Observables vs** **Subjects vs Replay Subjects vs Behavior Subjects:**

Observables

They are cold: Code gets executed when they have at least a single observer.

Creates copy of data: Observable creates copy of data for each observer.

Uni-directional: Observer can not assign value to observable(origin/master).

Subject

They are hot: code gets executed and value gets broadcast even if there is no observer.

Shares data: Same data get shared between all observers.

bi-directional: Observer can assign value to observable(origin/master).

If are using using subject then you miss all the values that are broadcast before creation of observer. So here comes Replay Subject

ReplaySubject

They are hot: code gets executed and value get broadcast even if there is no observer.

Shares data: Same data get shared between all observers.

bi-directional: Observer can assign value to observable(origin/master). plus

Replay the message stream: No matter when you subscribe the replay subject you will receive all the broadcasted messages.

In subject and replay subject you can not set the initial value to observable. So here comes Behavioral Subject

BehaviorSubject

They are hot: code gets executed and value get broadcast even if there is no observer.

Shares data: Same data get shared between all observers.

bi-directional: Observer can assign value to observable(origin/master). plus

Replay the message stream: No matter when you subscribe the replay subject you will receive last emitted message right away.

You can set initial value: You can initialize the observable with default value.

**Shared module:**

Create module and under this module u can create shared components, pipes, directives and export in export section of shared module

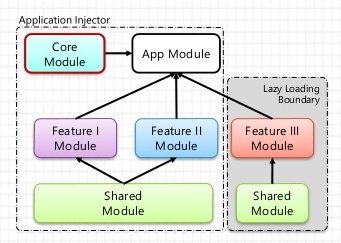
Then import this shared module in other modules so that you can access all from shared module in other module

https://medium.com/better-programming/angular-4-shared-modules-18ac50f24852

module import other module for reusing their components, pipes, directives etc.

**Core Module:**

Application wide model interface, interceptors, services , gaurds



<https://medium.com/@joao.aguas/angular-core-and-shared-modules-efe072bc9645>

**Pass data from one component to another as query string :**

Using Query Parameters with Router.navigate

goProducts() {

this.router.navigate(['/products'], { queryParams: { order: 'popular', 'price-range': 'not-cheap' } });

}

This will result in a URL that looks like this:

http://localhost:4200/products?order=popular&price-range=not-cheap

Using Query Parameters with RouterLink:

<a [routerLink]="['/products']" [queryParams]="{ order: 'popular'}">

Products

</a>

Accessing Query Parameter Values:

In order to retrieve the query parameters present in our application’s URL we can utilize ActivatedRoutes.

import { ActivatedRoute } from '@angular/router';

constructor(private route: ActivatedRoute) {}

this.route.queryParams.subscribe(params => {  
 this.queryParam = params[' order '];  
});

**View child vs view childen:**

@ViewChild and @ViewChildren are the types of decorators used to access and manipulate DOM elements, directives and components.

By using View child can access only one element / component

by using view children can access multiple components element / component

# what is difference between [annotation and decorator?](https://stackoverflow.com/questions/37317705/what-is-the-difference-between-annotation-and-decorator)

**Decorators**: A decorator is a function that adds metadata to a class, its members, or its method arguments.

Types of Decorators:

* Class decorators like @Component, @NgModule
* Property decorators like @Input and @Output
* Method decorators like @HostListener
* Parameter decorators like @Injectable

Traceur gives us annotations. TypeScript gives us decorators.

**Annotation** Annotations are a way to add metadata to class declarations.

RXJS:

<https://rxjs.dev/api?type=function>

some of important rxjs

| AREA | OPERATORS |
| --- | --- |
| Creation | from, fromEvent, of |
| Combination | combineLatest, concat, merge, startWith , withLatestFrom, zip |
| Filtering | debounceTime, distinctUntilChanged, filter, take, takeUntil |
| Transformation | bufferTime, concatMap, map, mergeMap, scan, switchMap |
| Utility | tap |
| Multicasting | share |

# Pipe, Subscribe, Map and Filter Examples with Angular 9/8:

<https://www.techiediaries.com/observable-pipe-subscribe-map-and-filter-examples/>

# RxJS switchMap, concatMap, mergeMap, exhaustMap:

https://morioh.com/p/fcfea9850b90

<https://stackblitz.com/edit/rxjs-higher-order-streams>

HTTP Example with Observables:

<https://codecraft.tv/courses/angular/http/http-with-observables/>

Observables gave us a new way of handling async requests. The other ways are promises, callbacks, and async/await.

<https://www.freecodecamp.org/news/what-are-observables-how-they-are-different-from-promises/>

https://jasonwatmore.com/post/2018/06/25/angular-6-communicating-between-components-with-observable-subject

An Observable is basically a function that can return a stream of values to an observer over time, this can either be synchronously or asynchronously. The data values returned can go from zero to an infinite range of values.