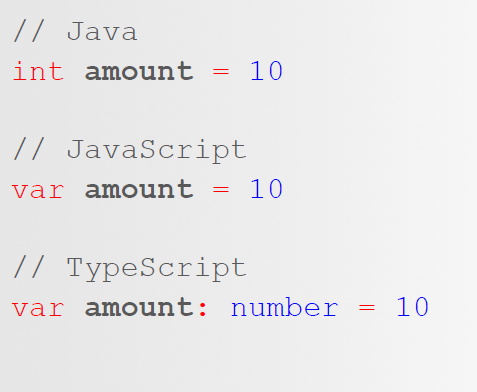
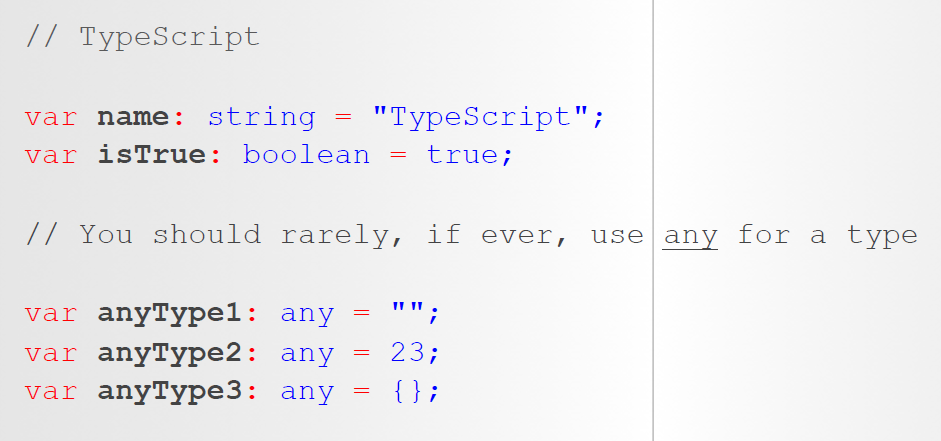
[Angular JS](http://angularjs.org/) is a client side JavaScript framework for data binding. etc..

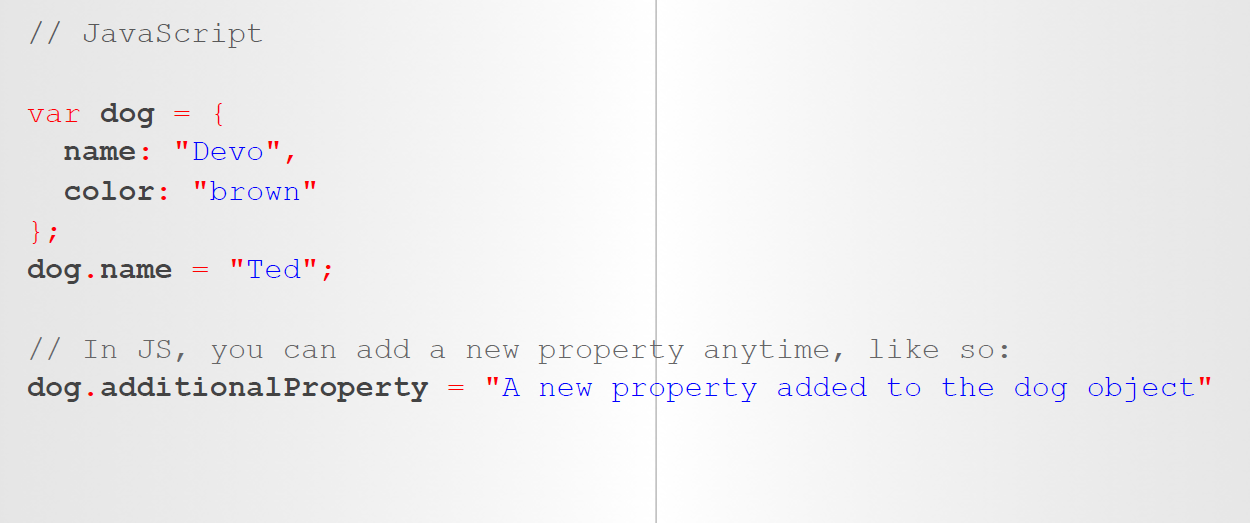
Typescript:

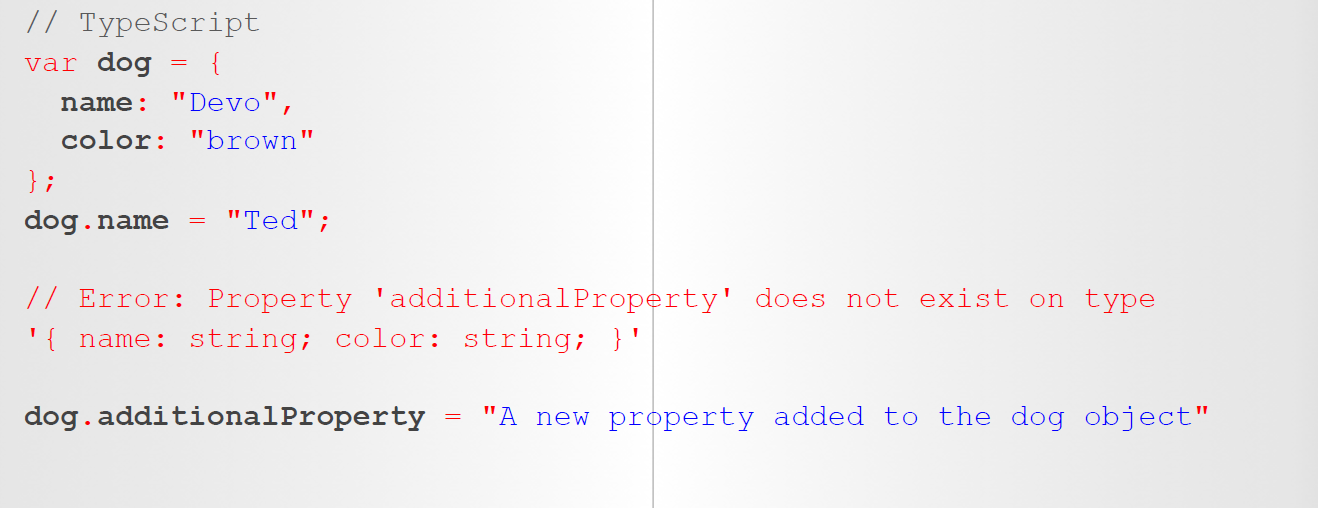
JS code can be written in Typescript

When datatypes are not mentioned while declaring, then the value has to be specified for the TS to implicitly know. Else the data type has to be specified explicitly.



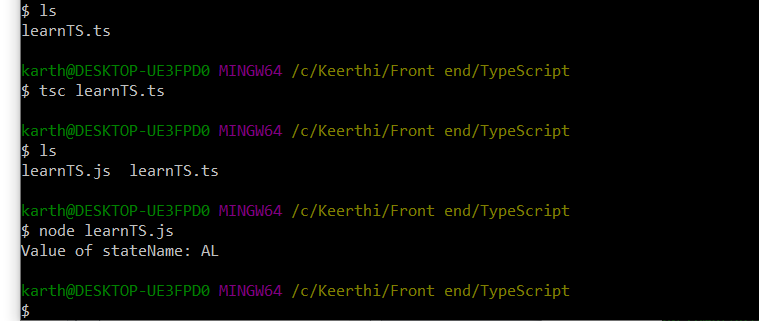






**How to compile and execute TS:**

Goto the folder where the TS file is…and do the below….tsc compiles the TS file and generates the JS file and JS file can be executed using node command



Object usage:

let objectAnimal = {

numberOfLegs : 4,

heightFeet : 2,

heightInches: 2,

weight: 23,

name: "Puppy"

};

Class members are public by default so members can be accessed outside the class

class Person{

age:number;

eyeColor:string;

hairColor:string;

isEmployed?:boolean; /\* Optional due to ? \*/

constructor(age : number, eyeColor:string, hairColor:string, isEmployed?:boolean){ /\* ? in the param tells that this param is an optional \*/

this.age = age;

this.eyeColor =eyeColor;

this.hairColor = hairColor;

if(isEmployed != undefined){ /\* if optional, if it is not passed, then it is undefined \*/

this.isEmployed = isEmployed;

}

}

}

const person1 = new Person(25, "Green", "Brown", true);

const person2 = new Person(34, "Brown", "Black");

console.log(person1);

console.log(person2);

console.log(person2.isEmployed); // property can be accessed but it will be undefined

person2.isEmployed =true;

console.log(person2); // now it will have the value

Class members are public by default so members can be accessed outside the class

const person1 = new Person(25, "Green", "Brown", true);

const person2 = new Person(34, "Brown", "Black");

console.log(person1);

console.log(person2);

console.log(person2.isEmployed); // property can be accessed but it will be undefined

person2.isEmployed =true;

console.log(person2); // now it will have the value

class Doctor extends Person{

constructor(age : number, eyeColor:string, hairColor:string, isEmployed?:boolean ){

super(age, eyeColor, hairColor, isEmployed ); /\* not ? is there in isEmployed since while accessing no question mark is neede...only needed when declaring

or function param is optional\*/

}

}

const doctor1 = new Doctor(23, "black", "brown");

const doctor2 = new Doctor(23, "black", "brown", true);

doctor2.age = 3; **/\* doctor2 is const...here the property value can be changed but we cannot do like doctor2 = new Doctor()**

console.log(doctor1);

console.log(doctor2);

Optional param in the function should always go at the end…

Required param should not follow optional param.

**Angular:**

<https://angular.io/guide/glossary>

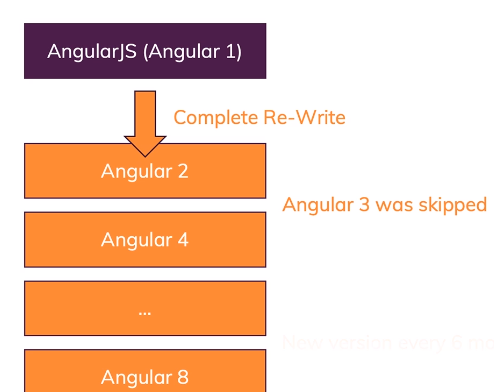
One class per file

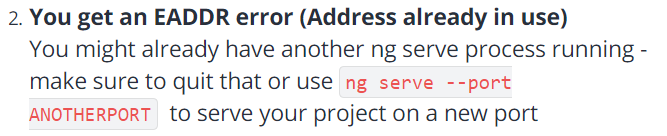
<ul class="list-group">

<li \*ngFor = "let t of tasks" class="list-group-item"> {{t}} </li>

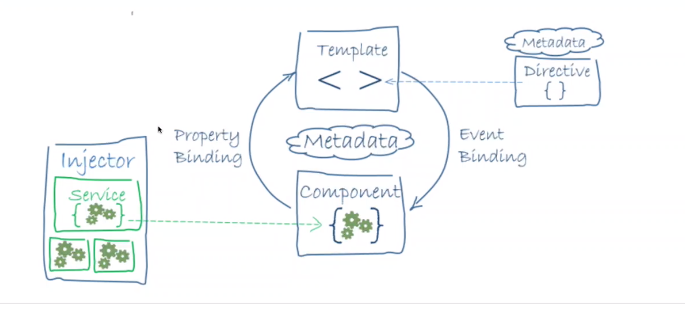
</ul>

Single html file and many javascript file









Template driven forms relies on html. When the form gets massive, html gets complex

<ul \*ngIf="displayList" class="list-group">

<li \*ngFor = "let t of tasks" class="list-group-item"> {{t}} </li>

</ul>

<button (click)="onToggleDisplay()">click</button>

<button \*ngIf="displayButton" (click)= "displayButton=!displayButton">Click</button>

\*ngIf, \*ngFor --- star decorators

One star decorator per tag

<button \*ngIf="buttonToggleMouseOver" (mouseover) = "buttonToggleMouseOver=!buttonToggleMouseOver">Mouse over</button>

<img [src]="image">

OnInit, AfterViewInit,OnDestroy – they are the interfaces and have to implement the following methods respectively : ngOnInit, ngAfterViewInit, ngOnDestroy

export class ListComponent implements OnInit, AfterViewInit,OnDestroy {

ngOnInit() {

console.log("In ngOnInit()");

}

ngAfterViewInit(){

console.log("In ngAfterViewInit()");

}

ngOnDestroy(){

console.log("In ngOnDestroy");

}

}

## Component lifecycle hooks overview:

Directive and component instances have a lifecycle as Angular creates, updates, and destroys them. Developers can tap into key moments in that lifecycle by implementing one or more of the lifecycle hook interfaces in the Angular core library.

Each interface has a single hook method whose name is the interface name prefixed with ng. For example, the [OnInit](https://angular.io/api/core/OnInit) interface has a hook method named ngOnInit() that Angular calls shortly after creating the component:

## <https://angular.io/guide/lifecycle-hooks>

To add input textbox in html, the below is added in **app.module.ts:**

import { FormsModule} from '@angular/forms'

## and the below in

imports: [

BrowserModule,

AppRoutingModule,

FormsModule

],

## Service:

## How to generate service:

## In cmd line under the project folder:

## $ ng generate service tasks

## CREATE src/app/tasks.service.spec.ts (328 bytes)

## CREATE src/app/tasks.service.ts (134 bytes)

## The above service files are generated. Ts class file will look like below:

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

@Injectable({

providedIn: 'root' // this tells that this service is avbl to all components

})

export class **TasksService** {

constructor() { }

}

## TasksService is the class name of the service that needs to be used in the components

## In List component TS class file, import the service file:

* 1. import {TasksService} from '../tasks.service'
  2. Add it to the constructor

constructor(**private** tasksService : TasksService) {

}

## Forms- Template based & Reactive

## New Application Procedure:

## Create the project: ng new <eCommerceApp>(say y to Angular routing and select scss – this is the sass file to style)

## Cd to that <eCommerceApp>

## Create components: ng generate component <adminComponent>

## Create components: ng generate component <ItemMasteComponent>

## 133 ng g service ItemsService

## 134 ng g c Admin

## 135 ng g c Navbar

## 136 ng g c Items-master

## 137 ng g c Items-master/Item-detail

## 138 ng g s Items

## 139 cd src/app/

## 140 ls

## Add bootstrap file (bootsrap.min.css)to src/assets folder and link that file in index.html

## Add the tags for the components in the app.component.html

## Add the item class in the Itemsservice :

## All optional params comes at last

export class Items{

name:string;

itemId:string;

description?:string;

imageUrl?: string;

price?:number;

constructor(name : string, description?: string, imageUrl?:string, price?:number){

this.name = name;

this.itemId = uuid.v4();

this.description = description;

this.imageUrl = imageUrl;

this.price = price;

}

}

## We want to generate unique identifier for the Item id so we need to add the package uuid. Add using the below package:

## $ npm install uuid

## 

## Import the package to be used in the Item class using

import \* as uuid from 'uuid'; (item class is in task service file)

## We are generating unique identifier for the ItemId so use this inside the constructor when an item is created like above

constructor(name : string, description?: string, imageUrl?:string, price?:number){

this.name = name;

this.itemId = uuid.v4();

this.description = description;

this.imageUrl = imageUrl;

this.price = price;

}

## Create an array for items:

items: Item[]=[

new Item("Book", "Coding book", "assets/Bear.jpeg", 23.5),

new Item("Movie", "Coding book", "assets/DaVinciCode.jpeg", 10.5),

new Item("Movie Poster", "Rail Pic", "assets/Rail.jpg", 8.5)

];

## Add a function to add an item in the service class.

## addItem(name : string, description?: string, imageUrl?:string, price?:number){

## const newItem:Item = new Item(name , description, imageUrl, price); // use let / var/ const

## this.items.push(newItem);

## 

## console.log("Item added: ");

## console.log(newItem);

## }

## Creating a reactive form now

## In app.module, add the below

## If we import it in app.module, it can be used through out the other modules

import { ReactiveFormsModule} from '@angular/forms'

imports: [

BrowserModule,

AppRoutingModule,

ReactiveFormsModule

],

## In admin.ts,

1. import {FormBuilder, FormGroup, Validators} from '@angular/forms'
2. import {ItemsService, Item} from '../items.service'

**15.** addItemForm = this.fb.group({

name: [''],

description:[''],

price:[0],

imageUrl:['']

});

**16.**

1. constructor(private itemsService: ItemsService, private fb:FormBuilder) {
2. console.log("Printing Item form");
3. console.log(this.addItemForm);
4. }

**17.** items:Item[] =[]; creating an array of items

**18.** Creating the submit function when the form is submitted

onSubmit(){

console.log("After Submitting");

console.log(this.addItemForm);

const name = this.addItemForm.value.name;

const description = this.addItemForm.value.description;

const price = this.addItemForm.value.price;

const imageUrl = this.addItemForm.value.imageUrl;

this.itemsService.addItem(name,description,price, imageUrl);

this.addItemForm.reset(); // reseting the form after submitting

}

**19. In admin.html**

<h1>Admin</h1>

// addItemForm is the form group

<form [formGroup]="addItemForm" (ngSubmit) = "onSubmit()">

<input type="text" formControlName = "name" id = "nameItem" class="form-control" placeholder="Enter the item name"><br>

<input type="text" formControlName = "description" id="descItem" class="form-control" placeholder="Enter the item Description"><br>

<input type="number" formControlName = "price" class="form-control" id ="priceItem" placeholder="Enter the item price"><br>

<input type="text" formControlName = "imageUrl" id = "imageUrl" class="form-control" placeholder="Enter the item imageUrl"><br>

<button type="submit" class=" btn btn-primary">Submit</button>

</form>

**20.**

## Npm install moment

1. import \* as moment from 'moment'
2. <h1> Learning Pipes & moment</h1>
3. <hr>
4. UpperCase: {{name| uppercase}} <br>
5. Lowercase: {{name| lowercase}} <br>
6. Titlecase: {{name| titlecase}} <br>
7. <hr>
8. Number: {{num}} <br>
9. Currency: {{num | currency}} <br>
10. Decimal: {{num | number: '1.1-3'}} <br>
11. Decimal: {{num | number: '1.1-2'}}
12. <hr>
13. Decimal: {{num | number: '1.1-2'| currency}}
14. <hr>
15. {{today}}<br>
16. Short Date: {{today| date: 'short'}}<br>
17. Short Date: {{today| date: 'medium'}}<br>
18. ShortDate Date: {{today| date: 'shortDate'}}<br>
19. MediumDate Date: {{today| date: 'mediumDate'}}<br>
20. LongDate Date: {{today| date: 'longDate'}}<br>
21. Short time: {{today| date: 'shortTime'}}<br>
22. Medium time: {{today| date: 'mediumTime'}}<br>
23. Long time: {{today| date: 'longTime'}}<br>
24. Full time: {{today| date: 'fullTime'}}<br>
25. <hr>

## Custom pipes:

## Ng generate pipe multiply

## Pipe file called multiplePipe.ts & multiply-pipe.spec.ts are created

## In multiply.pipe.ts:

@Pipe({

name: 'multiply' --🡪This is the declarator we have to use

})

export class MultiplyPipe implements PipeTransform {

transform(value: number, numberToMultiply?: number): number {

let finalNumberToMultiply;

if(isNaN(numberToMultiply))

finalNumberToMultiply = 1;

else

finalNumberToMultiply = numberToMultiply;

return finalNumberToMultiply \* value;

}

}

value: number -- > this is what gets passed

## Interceptor:

## All api request goes from service to the API. There is something called Interceptor where we can make the request go from service to the Interceptor first and then goes to the api.

## This interceptor is there where common actions such as including user credentials as part of the api request before sending it out.

## When error is received as response from api, error is also handled here like showing the msg to the user.

## SO the request comes here and then goes to the api.

## And the response comes to Interceptor first from the api and then is sent to the service.

## So why are HTTP interceptors useful? There are many reasons, but one common use case is to automatically attach authentication information to requests.

## Create a file called http-error.interceptor.ts by right clicking under app folder

## 

## HttpInterceptor interface and provider in app.module.ts tells that all api should pass through Interceptor.

## Intercept is the interface function which handles the api request.this request is there in req. This is immutable. If you want to modify this, it cannot be done. So create a new variable, copy the contents of req into this variable and add additional details and pass it to the next handler.

## next: HttpHandler – tells that this is next place the request goes to after receiving it in Interceptor. If no next is there then the request is sent to api.

## Calling next.handle means that we are passing control to the next interceptor in the chain, if there is one.

* next is the http handler, of type HttpHandler. The handler has a handlemethod that returns our desired HttpEvent observable.
* return next.handle(req).do(evt => {
* if (evt instanceof HttpResponse) {
* console.log('---> status:', evt.status);
* console.log('---> filter:', req.params.get('filter'));
* }

## });

## <https://alligator.io/angular/httpclient-interceptors/>

To wire-up our interceptor, let’s provide it in the app module or a feature module using the HTTP\_INTERCEPTORS token:

## 

## app.module.ts

# Add the Interceptor to Providers

The interceptor needs to be added to the HTTP\_INTERCEPTORS array. This is done by making the existing HTTP\_INTERCEPTORS array use the new class we’ve created. Add this in the providersarray for our application’s module.

## 

## Package.json is a file that has all the dependencies for the application to run. If there is no node\_modules folder and package.json file is there then run npm install. It will install all the dependencies.

## Node\_modules is not usually added to source control since the file gets large and this is added in .gitignore file.

## 

## So when we run npm install after downloading the application, it creates the nodes\_module by installing all the dependencies

## This happens only when we download from source control / gitlab. But it is there when we create an application using ng new.

## Now, adding a popup like a modal incase of error in interceptor.

## Admin module:

## Admin Component – import

## Add admin-routing.module.ts(arm) to admin folder

## Import arm to admin module

## Import forms, reactive forms module

## Import bootstrap module – ngbmodule

## Declarations: add admincomponent

## Imports – ARM, forms module, reactive forms module, Ngbmodule

## In app-module- remove everything related to admin like components since all are added in admin module

## In ARM, do whatever related to admin and make the file ready like app-routing-module like add imports

## 

## Add routes for the admin.

## 

# ActivatedRoute

## Provides access to information about a route associated with a component that is loaded in an outlet. Use to traverse the [RouterState](https://angular.io/api/router/RouterState) tree and extract information from nodes.

<a [routerLink] = "['/produce']">Go Back</a>

## Takes you to that route for which is defined in routing module. Takes you to a different page.

## To send data to another component from another component:

<app-add-to-cart [item]="item"></app-add-to-cart>

## egssInNest = [7,2,6]

## 

## function sortEggsInNest(a, b) {

## return a > b ? -1 : b > a ? 1 : 0;

## }

## eggsInNest.sort((a, b) => a - b);

## To add an icon to the webpage:

<link rel="icon" type="image/x-icon" href="favicon.ico">

## To run the application in a different port

## ng serve --port 5000