Reference

Best practice to use column class in html

Always use:

<div class=’container’>

<div class=” row”>

<div class=” col-md-12”>

How to use ‘NgZone’ and what it will do.

* It’s make angular methods to avail outside of angular (it can used in pluggable app)

First import NgZone (it is available in angular/core package), call it in constructor (like: private zone: NgZone)

Routing (Navigation through one url to another)

Routing for page (to navigate from one page to another page)

{path: ‘router\_name’, component: ‘component\_name’ …}

Observable.forkJoin([

this.getData(“json\_data\_1 or api\_service\_1”),

this.getData(“json\_data\_2 or api\_service\_2”)

])

How to pass multiple query parameter with Activated Routes: **see below**

This.activatedRoute.queryParams.subscribe( params => {

const reason = params [‘reason’];

const contentkey = params [‘contentkey’];

const alertmsg = params [‘alertmsg’];

//Apply condition based on your requirement like: see below code

If(reason) { // if reason is there then pass reason to the parameter.

// write the condition

}

If (contentkey) {

// write the conditions.

}

});

Way to access the object without dot notation in javascript.

1. We can access the object through dot(.) notation.
2. We can have the another way to access the object through square brackets([]).

Way to access the object:

// Given:

var foo = {'bar': 'baz'};

// Then

var x = foo['bar'];

// vs.

var x = foo.bar;

**Square bracket notation allows the use of characters that can't be used with dot notation:**

var foo = myForm.foo[]; // incorrect syntax

var foo = myForm["foo[]"]; // correct syntax

**Secondly, square bracket notation is useful when dealing with property names which vary in a predictable way:**

for (var i = 0; i < 10; i++) {

someFunction(myForm["myControlNumber" + i]);

}

**Roundup:**

* Dot notation is faster to write and clearer to read.
* Square bracket notation allows access to properties containing special characters and selection of properties using variables

**The bracket notation allows you to access properties by name stored in a variable:**

var obj = { "abc" : "hello" };

var x = "abc";

var y = obj[x];

console.log(y); //output - hello

obj.x would not work in this case.

<http://abusanad.net/2016/08/18/angular-2-cross-modules-communication-eventaggregator-overview/>

---- Reference link for Event Aggregator service.

<https://blog.thecodecampus.de/angular-2-use-hostbindings-set-class/> ---🡪 Reference link for Host Binding in angular2

**@HostBinding()** : The @HostBinding() annotation provides the range of possibilities. The advantage is, the required logic remains in the components class and is not set in the Template File. This makes it easier to test and improves the readability of the template file.

* It is the only way to set the CSS class to the host element within the component.
* To set the class use the @HostBinding() annotation either on the Boolean property of the class or use a method returning a Boolean. The value determines whether the class is set or not if the value/return is true, the class will be applied to the component DOM Element.

EX: **Host Binding with Method** EX: **Host Binding with a Property**

Export class class\_Name implements OnInit {

@HostBinding(‘class.isActive’) isActive: boolean;

Constructor() {}

ngOnInit() {}

}

Export class class\_Name implements OnInit {

Constructor() {}

ngOnInit() {}

isActive: boolean;

@HostBinding(‘class.isActive’) isActiveiAsMethod() {

return this.isActive;

}

}

The class to be set is specified within the round brackets, prefix by class. --- it is also possible to specify several classes by combining multiple annotations.

EX: @HostBinding(‘class.isActive’)

@HostBinding(‘class.current’) isActive: boolean;

====================================================================================================================================================

Reference : How to work with Reactive Form in angular 2:

<https://alligator.io/angular/reactive-forms-formarray-dynamic-fields/>

<https://alligator.io/angular/hostbinding-hostlistener/>

Way to create route in angular 2

/\* import files…\*/

-----------

export const routes: Routes = [

{path: ‘routing\_page\_name\_Like- documents’, component: ‘component\_name’,},

{-------------------------}

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

]

@NgModule({

Imports: [RouterModule.forRoot(routes)],

Exports: [RouterModule]

})

export class AppRoutingMocule{ }

Best practice to use bootstrap classes in html.

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

<div class=’row’>

<div class=’col-xs-12 col-md-9’>

--------//component selector define here like : <app-component></app-component>

</div>

</div>

Unit test case for iframe.Content.Document or getElementById

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

Initialize one variable inside describe like:

describe {

let testDocument;

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

// block of codes

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

// Write below code inside beforeEach

testDocument = {‘contentDocument’ : ‘<html><body><p id=’iframe’>Test Data </p></body></html>}

// write below peace of code inside the it function.

spyOn(‘service\_Object\_Name’ : ‘Method\_Name\_from\_service’).and.returnValue(‘testDocument’);

}

**Session Time out (If the user is not doing any action on the current page- page is idle for some time then it’s should get idle session timed out).**

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

Write the below code in component:

/\*-------imports-----\*/

// implementing this functionality by using ng-idle.

Export class…..

public isSessionTimedOut: Boolean;

public currentURL: string;

public idleTimeDuration: number;

constructor( private idle: Idle, private router: Router, …){

this. idleTimeDuration = 10; // setting default time out is 10 second.

}

ngOnInit() {

this.isSessionTimedOut = false;

this.router.events.filter(e => e instaceof NavigationEnd)

.subscribe(

(e:NavigationEnd) => {

This.currentURL = e.url;

This.idle.setIdle(this.idleTimeDuration);

This.idle.setTimeout(10); //sets time out period as 10 sec after 10 sec user will be considered timed out.

// sets the default interrupts, in this case things like: click, scroll, touches to the document

This.idle.setInterrupts(DEFAULT\_INTERRUPTSOURCES);

This.idle.onIdleStart.subscribe(()=>{

this.isSessionTimedOut = true;

});

This.idle.onIdleEnd.subscribe(()=>{

// ----

});

This.idle.onTimeout.subscribe(()=>

this.isSessionTimedOut = false;

) }

)

}

In HTML:

Put ngIf statement for displaying the model window on page idle activity

<div \*ngIf = “isSessionTimedOut”>

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

-----------

</div>

---------------Reference from Net-------------

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

import {Idle, DEFAULT\_INTERRUPTSOURCES} from '@ng-idle/core';

import {Keepalive} from '@ng-idle/keepalive';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

idleState = 'Not started.';

timedOut = false;

lastPing?: Date = null;

constructor(private idle: Idle, private keepalive: Keepalive) {

// sets an idle timeout of 5 seconds, for testing purposes.

idle.setIdle(5);

// sets a timeout period of 5 seconds. after 10 seconds of inactivity, the user will be considered timed out.

idle.setTimeout(5);

// sets the default interrupts, in this case, things like clicks, scrolls, touches to the document

idle.setInterrupts(DEFAULT\_INTERRUPTSOURCES);

idle.onIdleEnd.subscribe(() => this.idleState = 'No longer idle.');

idle.onTimeout.subscribe(() => {

this.idleState = 'Timed out!';

this.timedOut = true;

});

idle.onIdleStart.subscribe(() => this.idleState = 'You\'ve gone idle!');

idle.onTimeoutWarning.subscribe((countdown) => this.idleState = 'You will time out in ' + countdown + ' seconds!');

// sets the ping interval to 15 seconds

keepalive.interval(15);

keepalive.onPing.subscribe(() => this.lastPing = new Date());

this.reset();

}

reset() {

this.idle.watch();

this.idleState = 'Started.';

this.timedOut = false;

}

}

<http://www.concretepage.com/angular-2/angular-2-routing-and-navigation-example>

<http://www.discoversdk.com/blog/writing-unit-tests-in-angular-2> ------- Reference for Unit testing

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

<https://stackoverflow.com/questions/42412110/angular-2-how-to-pass-an-array-to-the-router-using-queryparams>

<https://www.youtube.com/watch?v=M0X634LyQ1k&index=3&list=PLC3y8-rFHvwg5gEu2KF4sbGvpUqMRSBSW>

<https://www.w3schools.com/icons/tryit.asp?filename=tryicons_awesome_intro_larger>

<http://sass-lang.com/documentation/file.SASS_REFERENCE.html#Operations> ------ SASS reference

<https://www.code-sample.com/2016/06/angular-2-interview-questions-and.html> ----- interview Questions

<https://alligator.io/angular/route-guards/>

<https://alligator.io/angular/angular-router-child-routes/>

<https://toddmotto.com/angular-2-forms-reactive> ----- Reactive Form creation in angular 2

<https://semaphoreci.com/community/tutorials/testing-angular-2-http-services-with-jasmine> ---- unit test for Http

<http://api.jquery.com/category/miscellaneous/collection-manipulation/> ------- Jquery

<http://www.concretepage.com/angular-2/angular-2-input-and-output-example> ----- Angular 2 Input and Output example

<https://www.npmjs.com/package/angular-idle-service> ----- ngIdle service

<https://keyholesoftware.com/services/education/angular-2-course/>

<http://marclloyd.co.uk/uncategorized/spying-on-localstorage-in-angular-2-unit-tests/>

<https://www.wisdomjobs.com/e-university/angular-2-interview-questions.html> --- interview question

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

Way to Write Unit test case for component and service files:

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

/\*\* imports all required dependencies \*/

**function initComponent(): ComponentFixture<Component\_Name>{**

**const = fixture TestBed.createComponent(Component\_Name);**

**fixture.detectChanges();**

**return fixture;**

**}**

describe(‘Component\_Name’, ()=>{

// variable declaration like: let var\_name;

beforeEach(async( () => {

TestBed.configureTestingModule({

Imports: [ //Define all dependent Module like: RouterTestingModule, etc…],

declarations: [ //declare component and u can declare dependent component as well ],

providers: [ //declare all the service file name like: LoginService, LoaderService, CookiesService … ],

})

}));

beforeEach(inject([ LoaderService //dependent service injection], (LoaderService) => {

loaderService = LoaderService;

…………………..

……………………

})) ;

It (‘component should get created ’, () => {

fixture = initComponent();

Component = fixture.componentInstance;

Expect(compoent).toBeTruthy();

})

// u can write multiple it functions based on your component

// u can test your component function that are appear in your component by calling the component method through the component.

It (‘Method should get call and defined’, () => {

fixture = initComponent;

component = fixture.componentInstance;

spyOn(service\_Object Name like: loaderService, ‘setTitle’)

expect(component.getProviderUrl).toBeDefined(); 🡪 // will check component method is defined or not. We can call the component method by component instance.

Component.Method\_Name();

})

})

Way to write Unit test cases for service file in angular 2

====================================================================================================================================================================

/\* Imports all required dependencies\*/

describe(‘Service\_name’, ()=>{

// variable declaration like: let var\_name;

beforeEach(async( () => {

TestBed.configureTestingModule({

Imports: [ //Define all dependent Module like: RouterTestingModule, etc…],

providers: [ //declare service here like: LoginService, LoaderService … {

provider: Http, HttpService,

useFactory: function (backendInstance: CoonectionBackend, defaultOptions: BaseRequestOptions){

return new Http (backendInstance, defaultOptions ) ;

},

Deps: [MockBackend, BaseRequestOptions],

} ];

});

}));

beforeEach (inject ( [ LoginService, MockBackend ], (loginService: LoginService, mockBackend: MockBackend ) => {

service = loginService;

backend = mockBackend;

…………………..

……………………

})) ;

// define all response json data.

It (‘service should get created’, () => {

expect (service).toBeTruthy();

});

// To check service method (get service will call here)

It (‘should get data from the service request’, (done) => {

backend.connection.subscribe ((connection) => {

expect(connection.request.method).toBe(RequestMethod.Get);

expect(connection.request.url).toBe(‘URL’);

connection.mockRespond(new Response(new ResponseOptions( {

body: Service\_Data;

status: 200;

})));

})

// you can call service subscribe methods like below

service.getEntitementData().subscribe((entitlementResp) =>{

expect(entitlementResp).not.toBe(null);

expect(entitlementResp.\_body.userAttribute).not.toBe(null);

done();

}, done.fail );

});

})

<https://toddmotto.com/component-events-event-emitter-output-angular-2>

 canactivate route: <https://blog.thoughtram.io/angular/2016/07/18/guards-in-angular-2.html>

Angular Cli config : <https://github.com/angular/angular-cli/wiki/angular-cli>

another one : <https://github.com/angular/angular-cli/Wiki>

Basic installation Steps : <https://github.com/angular/angular-cli/issues/2843> & <https://www.npmjs.com/package/angular-cli> & <https://angular.io/guide/quickstart>

Data flow : <http://www.sparkbit.pl/data-flow-angular-2-applications/> & <https://angular.io/guide/architecture>

Npm install : <https://docs.npmjs.com/all> & <https://docs.npmjs.com/cli/install>

Npm init command : It will create a package.json file where it will include the most common item.

To download latest npm package: npm -g install npm@latest

Anuglar 2 Life Cycle hooks : <https://angular.io/guide/lifecycle-hooks>

Angular 2 Reactive Forms Example: <https://embed.plnkr.co/?show=preview>

1. what is difference b/w <span> and <div> ?

2. what is position properties in CSS and explain difference type of position ?

3. How we can host the variable in javascript(variable Hosting in JS)?

**Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.**

Ans: We expected the result of the log to be: ReferenceError: hoist is not defined, but instead, its output is undefined. JavaScript has hoisted the variable declaration. This is what the code above looks like to the i

1. console.log(hoist); //output: undefined
2. var hoist = “The variables have been hoisted.”

JavaScript has hoisted the variable declaration. This is what the code above looks like to the interpreter: 1. Var hoist;

2. console.log(hoist);

3. hoist = “The variables have been hoisted.”

Eg: Reference Error : b is not defined.

Function hoist() {

a= 20;

Var b = 100;

}

Console.log(a)// print output as it was defined globally by javascript

Console.log(b)// give referenceerror: b is not defined.

Hoisting can be avoided if we use “use strict” mode at the top of the javascript.

JavaScript known as strict-mode, we can be more careful about how we declare our variables. By enabling [*strict mode*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode), we opt into a restricted variant of JavaScript that will not tolerate the usage of variables before they are declared.

<https://scotch.io/tutorials/understanding-hoisting-in-javascript>

**Let** are block scoped and not function scoped. we should *declare* then *assign* our variables to a value before using them.

The **const** keyword was introduced in es6 to allow *immutable variables*.

4. What is diff b/w var and let?

Ans The scope of a variable defined with var is function scope or declared outside any function, global.The scope of a variable defined with let is block scope.

The difference is scoping. **var** is scoped to the nearest function block and **let** is scoped to the nearest *enclosing* block, which can be smaller than a function block. Both are global if outside any block.

Let in the loop can re-binds it to each iteration of the loop, making sure to re-assign it the value from the end of the previous loop iteration, so it can be used to avoid issue with closures.

<http://www.jstips.co/en/javascript/keyword-var-vs-let/>

5. How to read JSON data in angular2?

6. what is Arrow function in angular2 and how we can use it explain with syntax ?

<https://ng2.codecraft.tv/es6-typescript/arrow/>

Arrow functions capture the this where the function is created rather than where it is invoked:

7. Difference Between normal function and arrow function in angular2

8. Inheritance in javascripts.

<https://stackoverflow.com/questions/16261044/why-javascript-does-not-support-inheritance-by-default>

9. What is promise in javascript. Differentiate it with observable.

JavaScript functions can be loosely classified as the following:

1. Function declarations: These are of the following form and are hoisted completely to the top. Now, we can understand why JavaScript enable us to invoke a function seemingly before declaring it.

hoisted();

Function hoisted() { console.log(“The function declaration has been hoisted”); }

1. Function expressions: Function expressions, however are not hoisted.

expression();

var expression = function() { console.log(“This function expression will not hoisted”) }

Order of precedence: It's important to keep a few things in mind when declaring JavaScript functions and variables.

1. Variable assignment takes precedence over function declaration

Var double = 22;

Function double(num) { return (num\*2) }

Console.log( typeod double ); // Output : number

1. Function declarations take precedence over variable declarations

Var double;

Function double(num) { return (num\*2) }

Console.log( typeod double ); // Output : function

12. What is the diff b/w ng-if and ng-hide?

<https://stackoverflow.com/questions/19177732/what-is-the-difference-between-ng-if-and-ng-show-ng-hide>

13. What is the diff b/w ng-bind and ng-model?

14. two-way binding in angular 1 how we can implement it?

15. predefined directives in angular1?

16. How to read query data from the data base?

17. what is @Input and @Output in angular2?

18. what is use of $scope in angular1?

19. what is promise how u can read the huge data from the backend?

20. difference b/w promise and observable?

Ans: **Promise**:A Promise handles a **single event** when an async operation completes or fails.

**Observable:** An Observable is like a Stream (in many languages) and allows to pass zero or more events where the callback is called for each event.

Often Observable is preferred over Promise because it provides the features of Promise and more. With Observable it doesn't matter if you want to handle 0, 1, or multiple events. You can utilize the same API in each case.

Observable also has the advantage over Promise to be **cancelable**. If the result of an HTTP request to a server or some other expensive async operation isn't needed anymore, the Subscription of an Observable allows to cancel the subscription, while a Promise will eventually call the success or failed callback even when you don't need the notification or the result it provides anymore.

Observable provides **operators** like map, forEach, reduce, ... similar to an array

21. have u developed any directive?

22. what is root directive in angular 1, how u will initialize the angular1?

23. what is the use of ng-repeat in angular1 explain?

<http://www.c-sharpcorner.com/article/top-50-angularjs-interview-questions-and-answers/>

24. how we can read the array of data through the service?

25. Diff between component and directivesss ?

<https://stackoverflow.com/questions/34613065/what-is-the-difference-between-component-and-directive>

26. What is Callback help in angularJs ?

27. what is difference between config and run in angularjs ?

28. what is data type in angular 2 ?

29. what is the replacement of $apply in angular2 ?

Ans: Zone.js (We have to import Ngzone for the same to use it )

30. Difference between synchronous and asynchronous in angularjs ?

Ans : To make sure the second calls are executed after the first one is finished, put the second call within then of the first call. To make multiple 'second' calls depending on the number of results of the first call, use $q.all

31. Promises in angularJs ?

<http://www.dwmkerr.com/promises-in-angularjs-the-definitive-guide/#whatarepromises>

<https://chariotsolutions.com/blog/post/angularjs-corner-using-promises-q-handle-asynchronous-calls/>

32. What makes the angular.copy() method so powerful?

A deep copy of a variable means it doesn’t point to the same memory reference as that variable. Usually assigning one variable to another creates a “shallow copy”, which makes the two variables point to the same memory reference. Therefore if one is changed, the other changes as well.

33. How would you make an Angular service return a promise? Write a code snippet as an example

Ans : To add promise functionality to a service, we inject the “$q” dependency in the service, and then use it .The $q library is a helper provider that implements promises and deferred objects to enable asynchronous functionality.

34. When creating a directive, it can be used in several different ways in the view. Which ways for using a directive do you know? How do you define the way your directive will be used?

Ans: When you create a directive, it can be used as an attribute, element or class name. To define which way to use, you need to set the restrict option in your directive declaration.The restrict option is typically set to: ‘A’ – only matches attribute name ‘E’ – only matches element name  
‘C’ – only matches class name, ‘AEC’ – matches either attribute or element or class name

Angular 2 Component Lifecycle:

A component has a lifecycle managed by Angular itself. Angular manages creation, rendering, data-bound properties etc. It also offers hooks that allow us to respond to key lifecycle events.

Here is the complete lifecycle hook interface inventory:

* ngOnChanges - called when an input binding value changes
* ngOnInit - after the first ngOnChanges
* ngDoCheck - after every run of change detection
* ngAfterContentInit - after component content initialized
* ngAfterContentChecked - after every check of component content
* ngAfterViewInit - after component's view(s) are initialized
* ngAfterViewChecked - after every check of a component's view(s)
* ngOnDestroy - just before the component is destroyed

The @ViewChild and @ViewChildren decorators provide access to the class of child component from the containing component.

The @ViewChild is a decorator function that takes the name of a component class as its input and finds its selector in the template of the containing component to bind to. @ViewChild can also be passed a template reference variable.

<https://angular.io/guide/component-interaction>

Remember that ngOnChanges method is called when there is a change on any Input property of a component

HTML DOM: The HTML DOM can be accessed with JavaScript (and with other programming languages).In the DOM, all HTML elements are defined as objects.

The programming interface is the properties and methods of each object.A property is a value that you can get or set (like changing the content of an HTML element).A method is an action you can do (like add or deleting an HTML element).

document.getElementById("demo").innerHTML = "Hello World!"; or

window.document.getElementById("demo").innerHTML = "Hello World!"; both are same. In the example above, getElementById is a method, while innerHTML is a property.

The document object represents your web page.If you want to access any element in an HTML page, you always start with accessing the document object.Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Element by Id: document.getElementById("intro");

## Finding HTML Elements by Tag Name: document.getElementsByTagName("p")

## Finding HTML Elements by Class Name: document.getElementsByClassName("intro");

## Finding HTML Elements by CSS Selectors: document.querySelectorAll("p.intro");

## Changing HTML Elements:

|  |  |
| --- | --- |
| Method | Description |
| element.innerHTML =  new html content | Change the inner HTML of an element |
| element.attribute = new value | Change the attribute value of an HTML element |
| element.setAttribute(attribute, value) | Change the attribute value of an HTML element |
| element.style.property = new style | Change the style of an HTML element |

Adding & Deleting Elements:

|  |  |
| --- | --- |
| Method | Description |
| document.createElement(*element*) | Create an HTML element |
| document.removeChild(*element*) | Remove an HTML element |
| document.appendChild(*element*) | Add an HTML element |
| document.replaceChild(*element*) | Replace an HTML element |
| document.write(*text*) | Write into the HTML output stream |
| document.getElementById(*id*).onclick = function(){*code*} | Adding event handler code to an onclick event |

Call a function on click event in javascript: object.onclick = function(){myScript};

Call a add event listener function in javascript: object.addEventListener("click", myScript);

<https://www.w3schools.com/angular/tryit.asp?filename=try_ng_filters_input> ( For searching item from the list of items through input field.

<https://www.w3schools.com/angular/tryit.asp?filename=try_ng_validate> (Validation on input fields and provide necessary error in front of it.

Following are the key differences between $apply() and $digest().

* Its use is to update the model properties forcibly.
* The $digest() method evaluates the watchers for the current scope. However, the $apply() method is used to evaluate watchers for root scope, that means it’s for all scopes.

$emit, $broadcast and $on:

Using $scope.$emit will fire an event up the $scope. Using $scope.$broadcast will fire an event down the $scope. Using $scope.$on is how we listen for these events.

### Explain $q service, deferred and promises.

* ‘Promises’ are post processing logics which are executed after some operation/action is completed whereas ‘deferred’ is used to control how and when those promise logics will execute.
* We can think about promises as “WHAT” we want to fire after an operation is completed while deferred controls “WHEN” and “HOW” those promises will execute.
* “$q” is the angular service which provides promises and deferred functionality.

# Explaing difference between Factory vs Service vs Provider: ?

# Ans : Factory : When you’re using a Factory you create an object, add properties to it, then return that same object. When you pass this service into your controller, those properties on the object will now be available in that controller through your factory.

Service: When you’re using Service, it’s instantiated with the ‘new’ keyword. Because of that, you’ll add properties to ‘this’ and the service will return ‘this’. When you pass the service into your controller, those properties on ‘this’ will now be available on that controller through your service.

Providers are the only service you can pass into your .config() function. Use a provider when you want to provide module-wide configuration for your service object before making it available.

**Scope in Javascipt:**

Lexical Scope : Whenever you see a function within another function, the inner function has access to the scope in the outer function, this is called Lexical Scope or Closure - also referred to as Static Scope.

Closure: A better example of how the *closure*side of things works, can be seen when returning a *function reference* - a more practical usage. Inside our scope, we can return things so that they’re available in the parent scope:

Eg: var sayHello = function (name) {  
 var text = 'Hello, ' + name;  
 return function () {  
 console.log(text);  
 };  
};

The closure concept we’ve used here makes our scope inside sayHello inaccessible to the public scope. Calling the function alone will do nothing as it *returns* a function:

sayHello('Todd'); // nothing happens, no errors, just silence...

The function returns a function, which means it needs assignment, and *then* calling:

var helloTodd = sayHello('Todd');  
helloTodd(); // will call the closure and log 'Hello, Todd'

Okay, I lied, you *can* call it, and you may have seen functions like this, but this will call your closure:

sayHello('Bob')(); // calls the returned function without assignment

AngularJS uses the above technique for its **$compile** method, where you pass the current scope reference into the closure:

$compile(template)(scope);

What is .call(), .apply() and .bind() function in javascript ?

We can use either .call () or .apply() to change the scope, but any further arguments are where the two differ: .call(scope, arg1, arg2, arg3) takes individual arguments, comma separated, whereas .apply(scope, [arg1, arg2]) takes an Array of arguments.

In order to understand this, you just need two more pieces of information. First, arguments is an Array-like object that represents all of the arguments passed into a function. Second, the apply method works exactly like the call primitive, except that it takes an Array-like object instead of listing the arguments out one at a time.

Our bind method simply returns a new function. When it is invoked, our new function simply invokes the original function that was passed in, setting the original value as this. It also passes through the arguments.

<http://blog.ninja-squad.com/2017/03/24/what-is-new-angular-4/> how angular2 different from angular4

What is optional parameter in angular2/4 ?

Ans: <https://www.typescriptlang.org/docs/handbook/functions.html>

**Angular from Scratch**:

**Decorator**: Angular has many decorators that attach metadata to classes so that it knows what those classes mean and how they should work.

**Template**:  A template is a form of HTML that tells Angular how to render the component.

**Metadata**: To tell Angular that HeroListComponent is a component, attach metadata to the class.In TypeScript, you attach metadata by using a decorator. Here's some metadata for HeroListComponent

Data binding plays an important role in communication between a template and its component. Data binding is also important for communication between parent and child components.

**Directives**: Angular templates are *dynamic*. When Angular renders them, it transforms the DOM according to the instructions given by directives. A directive is a class with a @[Directive](https://angular.io/api/core/Directive) decorator. A component is a directive-with-a-template; a @[Component](https://angular.io/api/core/Component) decorator is actually a @[Directive](https://angular.io/api/core/Directive) decorator extended with template-oriented features.

Two other kinds of directives exist: structural and attribute directives.

Structural directives alter layout by adding, removing, and replacing elements in DOM.(eg \*ngFor, \*ngIf)

Attribute directives alter the appearance or behavior of an existing element. In templates, they look like regular HTML attributes, hence the name. ( [(ngModel)], ngClass, ngStyle, ngSwitch )

*NgSwitch* is actually a set of three, cooperating directives: [NgSwitch](https://angular.io/api/common/NgSwitch), [NgSwitchCase](https://angular.io/api/common/NgSwitchCase), and [NgSwitchDefault](https://angular.io/api/common/NgSwitchDefault)

Where [NgSwitch](https://angular.io/api/common/NgSwitch) Is attribute directive without \* symbol. And Ngswitchcase and Ngswitchdefault are structural directive with \* symbol in the front.

**Dependency Injection**: *Dependency injection* is a way to supply a new instance of a class with the fully-formed dependencies it requires. Most dependencies are services. Angular uses dependency injection to provide new components with the services they need.

Template Reference Variable: A template reference variable is often a reference to a DOM element within a template. It can also be a reference to an Angular component or directive or a [web component](https://developer.mozilla.org/en-US/docs/Web/Web_Components). Use the hash symbol (#) to declare a reference variable.

The scope of a reference variable is the *entire template*. Do not define the same variable name more than once in the same template. The runtime value will be unpredictable. You can use the ref- prefix alternative to #. This example declares the fax variable as ref-fax instead of #fax.

Input & Output Property: An *Input* property is a *settable* property annotated with an @[Input](https://angular.io/api/core/Input) decorator. Values flow *into* the property when it is data bound with a [property binding](https://angular.io/guide/template-syntax#property-binding).

An *Output* property is an *observable* property annotated with an @[Output](https://angular.io/api/core/Output) decorator. The property almost always returns an Angular [EventEmitter](https://angular.io/api/core/EventEmitter). Values flow *out* of the component as events bound with an [event binding](https://angular.io/guide/template-syntax#event-binding)

**Guards**: A guard's return value controls the router's behavior: The guard can also tell the router to navigate elsewhere, effectively canceling the current navigation

* If it returns true, the navigation process continues.
* If it returns false, the navigation process stops and the user stays put.