**What is AngularJS ??**

AngularJS is a JavaScript-based open-source front-end web application framework.

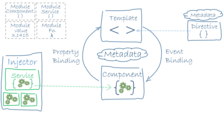
**What is Angular ??**

**Angular** is a TypeScript-based open-source front-end web application platform led by the **Angular** Team at Google.

Angular is a platform that makes it easy to build applications with the web. Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop.

Angular is mainly used to build single page web applications.

**Differences between Angular and AngularJS**

[](https://en.wikipedia.org/wiki/File:Architecture_of_an_Angular_2_application.png)

Architecture of an Angular application. The main building blocks are modules, components, templates, metadata, data binding, directives, services and dependency injection.

' Angular was a ground-up rewrite of AngularJS'***.***

* Angular does not have a concept of "scope" or controllers, instead it uses a hierarchy of components as its primary architectural characteristic.[[6]](https://en.wikipedia.org/wiki/Angular_(application_platform)#cite_note-6)
* Angular has a different expression syntax, focusing on "[ ]" for [property](https://en.wikipedia.org/wiki/Property_(programming)) binding, and "( )" for [event](https://en.wikipedia.org/wiki/Event_(computing)) binding[[7]](https://en.wikipedia.org/wiki/Angular_(application_platform)#cite_note-7)
* Modularity – much core functionality has moved to modules
* Angular recommends the use of Microsoft's [TypeScript](https://en.wikipedia.org/wiki/TypeScript) language, which introduces the following features:
  + Class-based [Object Oriented Programming](https://en.wikipedia.org/wiki/Object-oriented_programming)
  + [Static Typing](https://en.wikipedia.org/wiki/Static_typing)
  + [Generics](https://en.wikipedia.org/wiki/Generic_programming)
* [TypeScript](https://en.wikipedia.org/wiki/TypeScript) is a superset of [ECMAScript 6](https://en.wikipedia.org/wiki/ECMAScript_6) (ES6), and is [backwards compatible](https://en.wikipedia.org/wiki/Backward_compatibility) with [ECMAScript 5](https://en.wikipedia.org/wiki/ECMAScript_5) (i.e.: JavaScript). Angular also includes [ES6](https://en.wikipedia.org/wiki/ECMAScript_6):
  + [Lambdas](https://en.wikipedia.org/wiki/Lambda_(programming))
  + [Iterators](https://en.wikipedia.org/wiki/Iterator)
  + For/Of loops
  + [Python](https://en.wikipedia.org/wiki/Python_(programming_language))-style generators
  + [Reflection](https://en.wikipedia.org/wiki/Reflection_(programming))
* [Dynamic loading](https://en.wikipedia.org/wiki/Dynamic_loading)
* Asynchronous template compilation
* Iterative callbacks provided by RxJS. RxJS limits state visibility and debugging, but these can be solved with reactive add-ons like ngReact or ngrx.

**Single page applications ??**

A **single-page application** (**SPA**) is a [web application](https://en.wikipedia.org/wiki/Web_application) or [web site](https://en.wikipedia.org/wiki/Web_site) that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server.

**Component and Directive difference??**

*“Directives allow you to attach behavior to elements in the DOM.”*

 However in Angular 2, directives are split into the following 3 categories:

* Attribute
* Structural
* and… *Component*.

 Yes, in Angular 2, **Components are a type of Directive**.For further confirmation on this, let’s look at the API documentation for [@Component](https://angular.io/docs/ts/latest/api/core/index/Component-decorator.html), wherewe see the following description:

*“Angular components are a subset**of directives. Unlike directives, components always have a template and only one component can be instantiated per an element in a template.”*

 So in closing we can now say this:

**Directives in Angular 2 are:** The mechanism by which we attach behaviour to elements in the DOM, consisting of Structural, Attribute and Component types.

**Components in Angular 2 are:** The specific type of directive that allows us to utilize web component functionality - encapsulated, reusable elements available throughout our application.

**Change Detection**

Change Detection means updating the DOM whenever data is changed. Angular provides two strategies for Change Detection.

**https://dzone.com/articles/how-to-use-change-detection-in-angular**

**https://blog.angularindepth.com/everything-you-need-to-know-about-change-detection-in-angular-8006c51d206f**

**Life Cycle HOOKS**

**constructor**

This is invoked when Angular creates a component or directive by calling new on the class.

**ngOnChanges**

Invoked **every** time there is a change in one of the input properties of the component.

**ngOnInit**

Invoked when given component has been initialized.  
This hook is only called **once** after the first ngOnChanges

**ngDoCheck**

Invoked when the change detector of the given component is invoked. It allows us to implement our own change detection algorithm for the given component.

#### Important

ngDoCheck and ngOnChanges should not be implemented together on the same component.

**ngOnDestroy**

This method will be invoked just before Angular destroys the component.  
Use this hook to unsubscribe observables and detach event handlers to avoid memory leaks.

[**Hooks for the components children**](https://codecraft.tv/courses/angular/components/lifecycle-hooks/#_hooks_for_the_components_children)

These hooks are only called for components and not directives.

**ngAfterContentInit**

Invoked after Angular performs any content projection into the components view (see the previous lecture on Content Projection for more info).

**ngAfterContentChecked**

Invoked each time the content of the given component has been checked by the change detection mechanism of Angular.

**ngAfterViewInit**

Invoked when the component’s view has been fully initialized.

**ngAfterViewChecked**

Invoked each time the view of the given component has been checked by the change detection mechanism of Angular.

<https://codecraft.tv/courses/angular/components/lifecycle-hooks/>

**Data Binding**

data binding coordinated the communication between a component class and the template that its associated with.

4 different types of Data Binding in Angular:

* **Interpolation Binding** – dataflow from class to temp
* **Property Binding** – class to temp
* **Event Binding** – temp to class
* **Two-Way Binding** – property + event

<https://alligator.io/angular/data-binding-angular/>

**Routing**

The **Angular Router** enables navigation from one view to the next as users perform application tasks.

tells Angular where to load the components using a directive named router-outlet

**forRoot()** routing directory is used in root/app module to set “the route”.

**forChild()** routing directory is used in feature modules to set “the route”.

<https://toddmotto.com/angular-component-router>

<https://medium.com/all-is-web/https-medium-com-all-is-web-angular-5-routing-fad9b3648f67>

**LazyLoading**

**Lazy loading** is a technique in **Angular** that allows you to **load**  components asynchronously when a specific route is activated. This can add some initial performance during the initial **load**, especially if you have many components with complex routing.

[**https://codeburst.io/how-to-implement-lazy-loading-in-angular-6-419491102591**](https://codeburst.io/how-to-implement-lazy-loading-in-angular-6-419491102591)

**Auth Guards/Route Guards**

Angular’s route guards are interfaces which can tell the router **whether or not it should allow navigation to a requested route.** They make this decision by looking for a true or false return value from a class which implements the given guard interface.

There are five different types of guards and each of them is called in a particular sequence. The router’s behavior is modified differently depending on which guard is used. The guards are:

**CanActivate**

Checks to see if a user can visit a route.

**CanActivateChild**

Checks to see if a user can visit a routes children.

**CanDeactivate**

Checks to see if a user can exit a route.

**Resolve**

Performs route data retrieval before route activation.

**CanLoad**

Checks to see if a user can route to a module that lazy loaded.

* **CanActivate** - The canActivate method returns a boolean indicating **whether or not navigation to a route should be allowed.**

https://medium.com/@ryanchenkie\_40935/angular-authentication-using-route-guards-bf7a4ca13ae3

* **CanActivateChild** – https://scotch.io/courses/routing-angular-2-applications/canactivate-and-canactivatechild
* **CanDeactivate -** CanDeactivate is an interface that is implemented by a class to create a guard which decides if a route can be deactivated.

CanDeactivate guard can be used in the scenario, for example, suppose a user is changing form data and before saving, user tries to navigate away. In this scenario we can use CanDeactivate guard which will deactivate the route and open a Dialog Box to take user confirmation.

* **CanLoad –** The canActivate guard still allows the component for a given route to be activated (but not navigated to). If we wanted to prevent activation altogether, we could use the canLoad guard.
* **Resolve –** [**https://codeburst.io/understanding-resolvers-in-angular-736e9db71267**](https://codeburst.io/understanding-resolvers-in-angular-736e9db71267)

<https://codeburst.io/using-angular-route-guard-for-securing-routes-eabf5b86b4d1>

**Services**

* **Reusable code**

An **angular service** is simply a function that allows you to access its' **defined** properties and methods.

<https://www.quora.com/What-is-a-service-in-Angular-4>

**Singleton**

A service is singleton.

Angular instantiates the service object only once and all other components share the same instance.

There are two ways to make a service a singleton in Angular:

* Declare that the service should be provided in the application root.
* Include the service in the AppModule or in a module that is only imported by the AppModule.

**@Injectable**

We indicate this in service

It indicates a component or class has a dependency.

**Dependency Injection**

**Dependencies** are services or objects that a class needs to perform its function.

When a component is dependent on another component the dependency is injected or provided during the run time.

<https://toddmotto.com/angular-dependency-injection>

**crud operations**

**1.** CREATE operation using Http.post method.   
2. READ operation using Http.get method.   
3. UPDATE operation using Http.put method.   
**4.** DELETE operation using Http.delete method.

[**https://www.concretepage.com/angular-2/angular-4-crud-example**](https://www.concretepage.com/angular-2/angular-4-crud-example)

**Typescript**

**TypeScript** is an [open-source](https://en.wikipedia.org/wiki/Open-source_software) [programming language](https://en.wikipedia.org/wiki/Programming_language) developed and maintained by [Microsoft](https://en.wikipedia.org/wiki/Microsoft). It is a strict syntactical [superset](https://en.wikipedia.org/wiki/Superset) of [JavaScript](https://en.wikipedia.org/wiki/JavaScript), and adds optional static typing to the language.

TypeScript is designed for development of large applications and [transcompiles](https://en.wikipedia.org/wiki/Source-to-source_compiler) to JavaScript.[[5]](https://en.wikipedia.org/wiki/TypeScript#cite_note-5) As TypeScript is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs. TypeScript may be used to develop JavaScript applications for both [client-side](https://en.wikipedia.org/wiki/Client-side) and [server-side](https://en.wikipedia.org/wiki/Server-side) ([Node.js](https://en.wikipedia.org/wiki/Node.js)) execution.

TypeScript is an extension (a “superset”) of the JavaScript language. It differentiates itself from competitors like [CoffeeScript](http://coffeescript.org/) and [Dart](https://www.dartlang.org/) in that plain JavaScript code can be intermixed with TypeScript. Therefore JavaScript is TypeScript.

But TypeScript has to be compiled into JavaScript before it can run in any JavaScript engine (with a web browser or in [node.js](https://nodejs.org/en/)). This means you cannot embed TypeScript into a web page directly using <script> tags, but TypeScript (in .ts files) can get compiled into JavaScript (in .js files) for usage.

## TypeScript is Typed

Due to the dynamic nature of JavaScript variables, it can be difficult to identify bugs until the code is executed at runtime. TypeScript helps to solve this by providing optional static typing. The difference between dynamic and static typing can be huge when it comes to debugging an application. Take this overly simplistic scenario as an example:

var x = 2;

x = "two"; // some crazy person re-assigned a variable!

var y = x/2;

Now who would write such a thing? Well, if you wrote this with JavaScript, you may not spot the error until runtime. This code could be buried and you might not realize the error until a user complains.

TypeScript, by using type inference, would notice that you are changing the type from a number to a string. And you would know this **as you are typing**, not at runtime, saving you valuable debugging cycles.

TypeScript can also make your code more legible for other people. Take this chunk of JavaScript for instance:

function addThese(x, y) {

if (typeof x !== 'number' || typeof y !== 'number') {

throw new Error("Problemo!");

}

return x + y;

}

…could be simplified quite a bit in TypeScript with:

function addThese(x: number, y: number): number {

return x + y;

}

This is accomplished simply by adding types to your variables!

<https://developer.telerik.com/topics/web-development/what-is-typescript/>

**Datatypes**

Whenever a variable is created, the intention is to assign some value to that variable but what type of value can be assigned to that variable is dependent upon the datatype of that Variable.

| **BUILT-IN DATA TYPE** | **KEYWORD** | **DESCRIPTION** |
| --- | --- | --- |
| Number | number | It is used to represent both Integer as well as Floating-Point numbers |
| Boolean | boolean | Represents true and false |
| String | string | It is used to represent a sequence of characters |
| Void | void | Generally used on function return-types |
| Null | null | It is used when an object does not have any value |
| Undefined | undefined | Denotes value given to uninitialized variable |
| Any | any | If variable is declared with **any** data-type then any type of value can be assigned to that variable |

<https://www.geeksforgeeks.org/data-types-in-typescript/>

# Identifiers and Keywords in TypeScript

**Identifiers:** Identifiers are nothing but the names which is given to the members of any class like a variable, method name, class name, array name etc. Certain rules to be followed while declaring Identifiers:

* Identifier name can start with both upper-case as well as lower case letter but can’t start with numbers.
* Only \_ and $ symbols can be used for giving name to Identifiers, apart from these symbols, no other special symbol can be used.
* Keywords are different from Identifiers.
* Identifier are case sensitive and doesn’t contain spaces.

**Keywords:** Keywords are words which are responsible to perform some specific task or the words which represent some specific functionality.

**Difference between JavaScript and TypeScript**

* TypesScript is known as Object oriented programming language whereas JavaScript is a scripting language.
* TypeScript has a feature known as Static typing but JavaScript does not have this feature.
* TypeScript gives support for modules whereas JavaScript does not support modules.
* TypeScript has Interface but JavaScript does not have Interface.
* TypeScript support optional parameter function but JavaScript does not support optional parameter function.

**\*\*\***[**https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/**](https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/)

**Interface**

An interface is a syntactical contract that an entity should conform to. In other words, an interface defines the syntax that any entity must adhere to.

Interfaces define properties, methods, and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members. It often helps in providing a standard structure that the deriving classes would follow.

[**https://www.tutorialspoint.com/typescript/typescript\_interfaces.htm**](https://www.tutorialspoint.com/typescript/typescript_interfaces.htm)

**Generics**

*While basic types like interfaces are useful to describe data and basic functions signatures, generics helps making types “open” and reusable.*

[**https://medium.com/@wittydeveloper/typescript-generics-and-overloads-999679d121cf**](https://medium.com/@wittydeveloper/typescript-generics-and-overloads-999679d121cf)

**abstract class**

A **TypeScript Abstract class** is a **class** which may have some unimplemented methods. These methods are called **abstract** methods. We can't create an instance of an **abstract class**. But other **classes** can derived from **abstract class** and reuse the functionality of base **class**.

[**https://medium.com/@pagalvin/looking-at-abstract-classes-and-methods-in-typescript-9769de98f65b**](https://medium.com/@pagalvin/looking-at-abstract-classes-and-methods-in-typescript-9769de98f65b)

**enum**

Enums allow us to define a set of named constants. Using enums can make it easier to document intent, or create a set of distinct cases. TypeScript provides both numeric and string-based enums.

**https://www.typescriptlang.org/docs/handbook/enums.html**

[**https://medium.com/@katbusch/typescript-enums-explained-e5f9a101afc9**](https://medium.com/@katbusch/typescript-enums-explained-e5f9a101afc9)

**Typescript strict mode**

**Difference between observable and promise**

**Observable**

* Observables open up a continuous channel of communication in which multiple values of data can be emitted over time.
* An observable is like a stream and allows to pass zero or more events where the callback is called for each event.
* Observables are lazy. Function passed to Observable constructor gets called only when someone actually subscribes to an Observable: observable.subscribe();
* Observables allow you to resolve (or, as we say, “emit”) multiple values.

|  |
| --- |
| const numberObservable = new Observable((observer) => { |
|  | observer.next(5); |
|  | observer.next(10); |
|  | }); |
|  |  |
|  | numberObservable.subscribe(value => console.log(value)); |
|  | // prints 5 and 10 |

**Promise**

* Promise handles a single event when an async operation completes or fails.
* Promises are most commonly used to handle HTTP requests. In this model, you make a request and then wait for a single response. You can be sure that there won’t be multiple responses to the same request.
* Promise constructor immediately calls function passed to it. Promises are eager
* Promises actually enforce this semantics. You can create a Promise, which resolves with some value:

|  |
| --- |
| const numberPromise = new Promise((resolve) => { |
|  | resolve(5); |
|  | }); |
|  |  |
|  | numberPromise.then(value => console.log(value)); |
|  | // will simply print 5 |

But attempting to resolve Promise again with another value will fail. Promise is always resolved with the first value passed to the resolve function and ignores further calls to it:

|  |
| --- |
| const numberPromise = new Promise((resolve) => { |
|  | resolve(5); |
|  | resolve(10); |
|  | }); |
|  |  |
|  | numberPromise.then(value => console.log(value)); |
|  | // still prints only 5 |

**Change detection**