1. What do you understand by Angular expressions? How are Angular expressions different from JavaScript expressions?

Angular expressions are code snippets that are used to bind application data to HTML. Angular resolves the expressions, and the result is returned to where the expression is written. Angular expressions are usually written in double braces: {{ expression }} similar to JavaScript.

**Syntax:**

{{ expression }}

Following is a list of some differences between Angular expressions and JavaScript expressions:

1. The most crucial difference between Angular expressions and JavaScript expressions is that the Angular expressions allow us to write JavaScript in HTML. On the other hand, the JavaScript expressions don't allow.

2. The Angular expressions are evaluated against a local scope object. On the other hand, the JavaScript expressions are evaluated against the global window object. We can understand it better with an example. Suppose we have a component named test:

1. **import** { Component, OnInit } from '@angular/core';
2. @Component({
3. selector: 'app-test',
4. template: `
5. <h4>{{message}}</h4>,
6. styleUrls: ['./test.component.css']
7. })
8. export **class** TestComponent **implements** OnInit {
9. message:string = ?Hello world?;
10. constructor() { }
11. ngOnInit() {
12. }
13. }

In the above example, we can see that the Angular expression is used to display the message property. In the present template, we are using Angular expressions, so we cannot access a property outside its local scope (in this case, TestComponent). This proves that Angular expressions are always evaluated based on the scope object rather than the global object.

3. The Angular expressions can handle null and undefined, whereas JavaScript expressions cannot.

See the following JavaScript example:

1. <!DOCTYPE html**>**
2. **<html** lang="en"**>**
3. **<head>**
4. **<meta** charset="UTF-8"**>**
5. **<meta** name="viewport" content="width=device-width, initial-scale=1.0"**>**
6. **<title>**JavaScript Test**</title>**
7. **</head>**
8. **<body>**
9. **<div** id="foo"**><div>**
10. **</body>**
11. **<script>**
12. 'use strict';
13. let bar = {};
14. document.getElementById('foo').innerHTML = bar.x;
15. **</script>**
16. **</html>**

After running the above code, you see undefined displayed on the screen. Although it's not ideal to leave any property undefined, the user does not need to see this.

Now see the following Angular example:

1. **import** { Component, OnInit } from '@angular/core';
2. @Component({
3. selector: 'app-new',
4. template: `
5. <h4>{{message}}</h4>       `,
6. styleUrls: ['./new.component.css']
7. })
8. export **class** NewComponent **implements** OnInit {
9. message:object = {};
10. constructor() { }
11. ngOnInit() {
12. }
13. }

In the above example, you will not see undefined being displayed on the screen.

4. In Angular expressions, we cannot use loops, conditionals, and exceptions. The difference which makes Angular expressions quite beneficial is the use of pipes. Angular uses pipes (known as filters in AngularJS) to format data before displaying it.

See this example:

1. **import** { Component, OnInit } from '@angular/core';
2. @Component({
3. selector: 'app-new',
4. template: `
5. <h4>{{message | lowercase}}</h4>,
6. styleUrls: ['./new.component.css']
7. })
8. export **class** NewComponent **implements** OnInit {
9. message:string = "HELLO JAVATPOINT";
10. constructor() { }
11. ngOnInit() {
12. }
13. }

In the above example, we have used a predefined pipe called lowercase, which transforms all the letters in lowercase. If you run the above example, you will see the output displayed as "hello javatpoint".

On the other hand, JavaScript does not have the concept of pipes.

1. **Why is Angular preferred over other frameworks? / What are some advantages of Angular over other frameworks?**

Due to the following features, Angular is preferred over other frameworks:

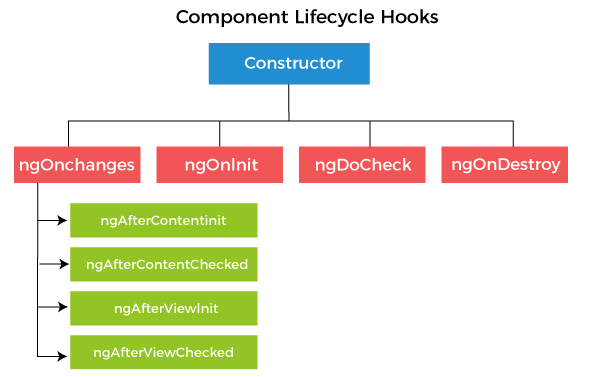
**Extraordinary Built-in Features:** Angular provides several out of the box built-in features like routing, state management, RxJS library, Dependency Injection, HTTP services, etc. That's why the developers do not need to look for the above-stated features separately.

**Declarative UI:** Angular has declarative UI. It uses HTML to render the UI of an application as it is a declarative language. It is much easier to use than JavaScript.

**Long-term Google Support:** Angular is developed and maintained by Google. Google has a long term plan to stick with Angular and provide support

1. **What** are the different Lifecycle hooks of Angular? Explain them in short.

When the Angular components are created, they enter their lifecycle and remain when they are destroyed. Angular Lifecycle hooks are used to check the phases and trigger changes at specific phases during the entire duration.



**ngOnChanges( ):** This method is called when one or more input properties of the component are changed. The hook receives a SimpleChanges object containing the previous and current values of the property.

**ngOnInit( ):** This is the second lifecycle hook. It is called once, after the ngOnChanges hook. It is used to initialize the component and sets the input properties of the component.

**ngDoCheck( ):** This hook is called after ngOnChanges and ngOnInit and is used to detect and act on changes that Angular cannot detect. In this hook, we can implement our change detection algorithm.

**ngAfterContentInit( ):** This hook is called after the first ngDoCheck hook. This hook responds after the content gets projected inside the component.

**ngAfterContentChecked( ):** This hook is called after ngAfterContentInit and every subsequent ngDoCheck. It responds after the projected content is checked.

**ngAfterViewInit( ):** This hook is called after a component's view or initializing a child component's view.

**ngAfterViewChecked( ):** This hook is called after ngAfterViewInit. It responds after the component's view or when the child component's view is checked.

**ngOnDestroy( ):** This hook is called just before Angular destroys the component. This is used to clean up the code and detach event handlers.

In the above hooks we have described, the ngOnInit hook is the most often used hook. Let's see how to use the ngOnInit hook. If you have to process a lot of data during component creation, it's better to do it inside the ngOnInit hook rather than the constructor:

**See the example:**

1. **import** { Component, OnInit } from '@angular/core';
2. @Component({
3. selector: 'app-test',
4. templateUrl: './test.component.html',
5. styleUrls: ['./test.component.css']
6. })
7. export **class** TestComponent **implements** OnInit {
8. constructor() { }
9. ngOnInit() {
10. **this**.processData();
11. }
12. processData(){
13. // Do something..
14. }
15. }

In the above code, you can see that we have imported OnInit, but we have used the ngOnInit function. This is how we can use the rest of the hooks as well.

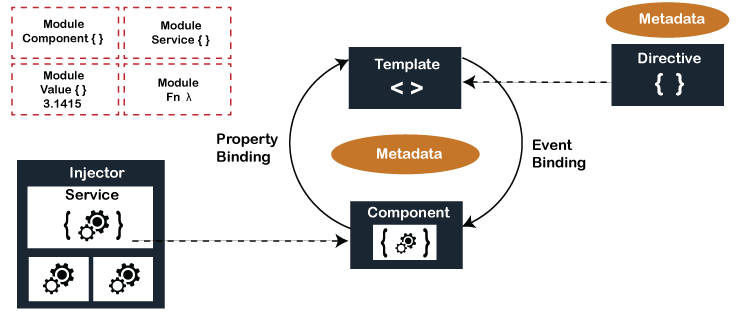
1. What is the main difference between JIT and AOT in Angular?

Following are the main differences between JIT and AOT compiler in Angular:

* Just-in-Time (JIT) compiler compiles our app in the browser at run-time while Ahead-of-Time (AOT) compiler is used to compile your app at build time on the server.
* The JIT compilation runs by default when you run the ng build (build only), or ng serve (build and serve locally) CLI commands. This is used for development. On the other hand, we have to include the --aot option with the ng build or ng serve command for AOT compilation.
* JIT and AOT are both two ways used to compile code in an Angular project. JIT compiler is used in development mode while AOT is used for production mode.
* JIT is easy to use. We can easily implement features and debug in JIT mode because here we have a map file while AOT does not. On the other hand, the biggest advantage of using AOT for production is that it reduces the bundle size for faster rendering.

1. What are the main building blocks of an Angular application? Explain with the pictorial diagram of Angular architecture.

Following are the main building blocks of an Angular application. You can see them in the following picture:



1. What is the difference between Observables and Promises in Angular?

In Angular, as soon as we make a promise, the execution takes place, but this is not the case with observables because they are lazy. It means nothing happens until a subscription is made.

|  |  |
| --- | --- |
| **Promise** | **Observable** |
| It emits a single value. | It emits multiple values over a period of time. |
| Not Lazy | Lazy. An observable is not called until we subscribe to the observable. |
| We can not cancel it. | We can cancel it by using the unsubscribe() method. |
|  | Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc. |

**Let's understand it by an example:**

1. const observable = rxjs.Observable.create(observer =**>** {
2. console.log('This is what inside an observable');
3. observer.next('Hello JavaTpoint');
4. observer.complete();
5. });
6. console.log('Before subscribing an Observable');
7. observable.subscribe((message)=**>** console.log(message));

When you run the above Observable, you can see the following messages displayed in the following order:

1. Before subscribing an Observable
2. This is what inside an observable
3. Hello JavaTpoint

Here, you can see that observables are lazy. Observable runs only when someone subscribes to them. That's why the message "Before subscribing an Observable" is displayed ahead of the message inside the observable.

**Now see the example of a Promise:**

1. const promise = new Promise((resolve, reject) =**>** {
2. console.log('This is what written inside promise');
3. resolve('Hello JavaTpoint');
4. });
5. console.log('Before calling then method on Promise');
6. greetingPoster.then(message =**>** console.log(message));

When you run the above Promise, you will see the messages displayed in the following order:

1. This is what written inside Promise
2. Before calling then method on Promise
3. Hello JavaTpoint

Here, you can see that the message inside Promise is displayed first. This means that the Promise runs first, and then the method is called.

The next difference between them is that Promises are always asynchronous; even when the Promise is immediately resolved. On the other hand, an Observable can be both synchronous and asynchronous.

In the case of the above example, observable is synchronous. Let's see the case where an observable can be asynchronous:

1. const observable = rxjs.Observable.create(observer =**>** {
2. setTimeout(()=**>**{
3. observer.next('Hello JavaTpoint');
4. observer.complete();
5. },3000)
6. });
7. console.log('Before calling subscribe on an Observable');
8. observable.subscribe((data)=**>** console.log(data));
9. console.log('After calling subscribe on an Observable');

When you run the above observable, you will see the messages in the following order:

1. Before calling subscribe on an Observable
2. After calling subscribe on an Observable
3. Hello JavaTpoint
4. What **are directives in Angular?**

A directive is a class in Angular that is declared with a @Directive decorator. Every directive has its own behavior, and you can import them into various components of an application.

1. What is Angular CLI?

Angular CLI is a short form for Angular Command Line Interface. It is a command-line interface to scaffold and build angular apps using node.js style modules.

To use Angular CLI, we have to install it by using the following **npm** command:

1. npm install @angular/cli@latest

**Following is a list of some useful commands which would be very helpful while creating angular projects:**

* **Creating New Project:** ng new
* **Generating Components, Directives & Services:** ng generate/g
* **Running the Project:** ng serve

1. What is lazy loading in Angular?

Lazy loading is one of the most powerful and useful concepts of Angular Routing. It makes the web pages easy to download by downloading them in chunks instead of downloading everything in a big bundle. Lazy loading facilitates asynchronously loading the feature module for routing whenever required using the property loadChildren.

See the following example where we are going to load both Employee and Order feature modules lazily.

**See the example:**

1. **const** routes: Routes = [
2. {
3. path: 'employees',
4. loadChildren: () => **import**('./employees/employees.module').then(module => module.EmployeesModule)
5. },
6. {
7. path: 'orders',
8. loadChildren: () => **import**('./orders/orders.module').then(module => module.OrdersModule)
9. },
10. {
11. path: '',
12. redirectTo: '',
13. pathMatch: 'full'
14. }
15. ];
16. **What is Angular Router?**

Angular Router is a mechanism that facilitates users to navigate from one view to the next as users perform application tasks. It follows the concept model of browser's application navigation.