# JavaScript Advance

<https://mindmajix.com/javascript-interview-questions>

<https://www.toptal.com/javascript/interview-questions>

<https://blog.risingstack.com/javascript-clean-coding-best-practices-node-js-at-scale/>

<https://www.codementor.io/nihantanu/21-essential-javascript-tech-interview-practice-questions-answers-du107p62z>

<https://www.thatjsdude.com/interview/js2.html>

<https://career.guru99.com/top-85-javascript-interview-questions/>

<http://www.techbeamers.com/javascript-interview-questions-answers/>

<https://www.web-technology-experts-notes.in/2015/10/object-oriented-javascript-interview-questions-and-answers-for-experienced.html>

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

<http://www.webgranth.com/javascript-interview-questions-answers-basic-advanced>

Iterate over javascript object

What are mutables in js

Push at 30 th position in array

Closure and variable hoisting with console

Diffrence between arrow and

ananoymus function

Can we stop bubbling

How to implement singleton class in javascript

## Important Definitions: ----------------------------------------------------

## JavaScript types?

* Number
* Object
* String
* Boolean
* Function
* Null
* Undefined

## Closure

 A closure is the combination of a function and the lexical environment within which that function was declared.

## Promise

**Promises are chainable concurrency primitive that acts as placeholder for a future value.**

## Basic of JavaScript: ---------------------------------------------------------

## What are JavaScript types?

* Number
* Object
* String
* Boolean
* Function
* Null
* Undefined

## What is negative infinity?

Negative Infinity is a number in JavaScript which can be derived by dividing negative number by zero.

## What are global variables? How are these variable declared and what are the problems associated with using them?

Global variables are those that are available throughout the length of the code, that is, these have no scope. The var keyword is used to declare a local variable or object. If the var keyword is omitted, a global variable is declared.

## Explain the working of timers in JavaScript? Also elucidate the drawbacks of using the timer, if any?

Timers are used to execute a piece of code at a set time or also to repeat the code in a given interval of time. This is done by using the functions **setTimeout, setInterval**and**clearInterval**.

The **setTimeout(function, delay)** function is used to start a timer that calls a particular function after the mentioned delay. The **setInterval(function, delay)** function is used to repeatedly execute the given function in the mentioned delay and only halts when cancelled. The **clearInterval(id)** function instructs the timer to stop.

Timers are operated within a single thread, and thus events might queue up, waiting to be executed.

## How can you convert the string of any base to integer in JavaScript?

The parseInt() function is used to convert numbers between different bases. parseInt() takes the string to be converted as its first parameter, and the second parameter is the base of the given string.

In order to convert 4F (of base 16) to integer, the code used will be –

|  |  |
| --- | --- |
| 1 | parseInt ("4F", 16); |

## What is the function of delete operator?

The functionality of delete operator is used to delete all variables and objects in a program but it cannot delete variables declared with VAR keyword.

## What is the data type of variables of in JavaScript?

All variables in the JavaScript are object data types.

## What is break and continue statements?

Break statement exits from the current loop.

Continue statement continues with next statement of the loop.

## I\*\*\*\*What is the use of type of operator?

‘Typeof’ is an operator which is used to return a string description of the type of a variable.

## What are the different types of errors in JavaScript?

There are three types of errors:

* **Load time errors**: Errors which come up when loading a web page like improper syntax errors are known as Load time errors and it generates the errors dynamically.
* **Run time errors**: Errors that come due to misuse of the command inside the HTML language.
* **Logical Errors**: These are the errors that occur due to the bad logic performed on a function which is having different operation.

## What is the use of Push method in JavaScript?

* The push method is used to add or append one or more elements to the end of an Array. Using this method, we can append multiple elements by passing multiple arguments

## What is unshift method in JavaScript?

* Unshift method is like push method which works at the beginning of the array.  This method is used to prepend one or more elements to the beginning of the array.

## How can a value be appended to an array?

* A value can be appended to an array in the given manner –
* arr[arr.length] = value;

## I\*\*\*\*What is the difference between .call() and .apply()?

* The function .call() and .apply() are very similar in their usage except a little difference. .call() is used when the number of the function’s arguments are known to the programmer, as they have to be mentioned as arguments in the call statement. On the other hand, .apply() is used when the number is not known. The function .apply() expects the argument to be an array.

## I\*\*\*\*What is difference between Var, let and const?

**Var**

The JavaScript variables statement is used to declare a variable and, optionally, we can initialize the value of that variable.

Example: var a =10;

Variable declarations are processed before the execution of the code.

The scope of a JavaScript variable declared with var is its current execution context.

The scope of a JavaScript variable declared outside the function is global.

Consider the following code snippet.

function nodeSimplified(){

var a =10;

console.log(a); // output 10

if(true){

var a=20;

console.log(a); // output 20

}

console.log(a); // output 20

}

In the above code, you can find, when the variable is updated inside the if loop, that the value of variable "a" updated 20 globally, hence outside the if loop the value persists. It is similar to the Global variable present in other languages. But, be sure to use this functionality with great care because there is the possibility of overriding an existing value.

**let**

The let statement declares a local variable in a block scope. It is similar to var, in that we can optionally initialize the variable.

Example: let a =10;

The let statement allows you to create a variable with the scope limited to the block on which it is used.

It is similar to the variable we declare in other languages like Java, .NET, etc.

Consider the following code snippet.

function nodeSimplified(){

let a =10;

console.log(a); // output 10

if(true){

let a=20;

console.log(a); // output 20

}

console.log(a); // output 10

}

It is almost the same behavior we see in most language.

function nodeSimplified(){

let a =10;

let a =20; //throws syntax error

console.log(a);

}

Error Message: Uncaught SyntaxError: Identifier 'a' has already been declared.

However, with var, it works fine.

function nodeSimplified(){

var a =10;

var a =20;

console.log(a); //output 20

}

The scope will be well maintained with a let statement and when using an inner function the let statement makes your code clean and clear.

I hope the above examples will help you better understand the var and let commands and if you have any queries please write me in the comment section.

**const**

const statement values can be assigned once and they cannot be reassigned. The scope of const statement works similar to let statements.

Example: const a =10;

function nodeSimplified(){

const MY\_VARIABLE =10;

console.log(MY\_VARIABLE); //output 10

}

As per usual, naming standards dictated that we declare the const variable in capital letters. const a =10 will work the same way as the code given above. Naming standards should be followed to maintain the code for the long run.

Question: What will happen when we try to reassign the const variable?

Consider the following code snippet.

function nodeSimplified(){

const MY\_VARIABLE =10;

console.log(MY\_VARIABLE); //output 10

MY\_VARIABLE =20; //throws type error

console.log(MY\_VARIABLE);

}

## Determine the output of the code below. Explain your answer.

A

console.log(0.1 + 0.2);

console.log(0.4 + 0.1 == 0.5);

This is a trick question in that at first glance, you might expect the console to print out “0.3” and “true.” The correct answer is that you can’t know for sure, because of how JavaScript treats floating point values. In fact, in the above example, it will print out:

0.30000000000000004

False

## Write a function that can determine whether a string is a palindrome in under 100 characters.

A palindrome is a word, phrase, or sequence of letters that reads the same backwards or forwards. It also makes a great test for checking their ability to handle strings.

function isPalindrome(str) {

str = str.replace(/s/g, '').toLowerCase();

return (str == str.split('').reverse().join(''));

}

## I\*\*\*\*How to create a class?

JavaScript does not have a class definition. To mimic classes in JavaScript functions can be used to declare a class.

Example:

Function Student (name .roll){

this.name = name;

this.roll = roll;

}

## How to create an object?

An object in JavaScript can be created using two was:

New Key word:

To create a student object from the above student class we can call the Student function using new keyword.

var student1 = new Student(‘santosh’,2)

Anonymous Object:

Anonymous objects can be created using pair of curls’ braces containing property name and value pairs.

Var rose = {‘color’: ‘red’}

## What is prototype property?

By Using Prototype we can add new members to an existing object. Every JavaScript object has this property internally. Initially it is an empty object.

Example:

function Student (name, roll){

this.name = name;

this.roll = roll;

}

var student1 = new Student(’sangeeta’,30);

Student.prototype.mark = 100;

Checkout the below chrome console for the use of Protype.

By using prototype a new property mark has been added to student object with a value of 100.

## I\*\*\*\*How to empty an array in JavaScript?

var arrayList = ['a','b','c','d','e','f'];

**Method 1**

arrayList = []

**Method 2\***

arrayList.length = 0;

**Method 3**

arrayList.splice(0, arrayList.length);

**Method 4**

while(arrayList.length){

arrayList.pop();

}

## When would you use var in your declaration and when you wouldn't?

Always use var. Not using var for variable declaration will traverse scopes all the way up till the global scope. If variable with that name is not found it will declare it in the global scope. Therefore not using var implicitly declares variable in the global scope (which, let me remind you, is a bad practice).

(function() {

baz = 5;

var bar = 10;

})();

console.log(baz); // outputs 5

//console.log(bar); // error: bar is not defined

## What are the differences between null and undefined?

JavaScript has two distinct values for nothing, null and undefined.

**undefined**

undefined means, value of the variable is not defined. JavaScript has a global variable undefined whose value is "undefined" and typeof undefined is also "undefined". Remember, undefined is not a constant or a keyword. undefined is a type with exactly one value: undefined. Assigning a new value to it does not change the value of the type undefined.

**8 Ways to get Undefined:**

A declared variable without assigning any value to it.

Implicit returns of functions due to missing return statements.

return statements that do not explicitly return anything.

Lookups of non-existent properties in an object.

Function parameters that have not passed.

Anything that has been set to the value of undefined.

Any expression in the form of void(expression)

The value of the global variable undefined

**null**

null means empty or non-existent value which is used by programmers to indicate “no value”. null is a primitive value and you can assign null to any variable. null is not an object, it is a primitive value. For example, you cannot add properties to it. Sometimes people wrongly assume that it is an object, because typeof null returns "object".

## What the difference is between == and ===? Which one would you use?

The equality (==) operator will compare for equality after doing necessary type casting, the identity operator (===) doesn't do any conversions. A good practice suggested by Douglas Crockford is to always use strict equality, couple of examples from Douglas' book JavaScript: The Good Parts

'' == '0' // false

0 == '' // true

0 == '0' // true

false == 'false' // false

false == '0' // true

false == undefined // false

false == null // false

null == undefined // true

## What are the differences between == and ===?

The simplest way of saying that, == will not check types and === will check whether both sides are of same type. So, == is tolerant. But under the hood it converts to its convenient type to have both in same type and then do the comparison.

=== compares the types and values. Hence, if both sides are not same type, answer is always false. For example, if you are comparing two strings, they must have identical character sets. For other primitives (number, boolean) must share the same value.

Rule for implicit coercion: Comparison by using == does implicit type conversion under the hood. And rules for implicit coercion are as follows-

If both operands are same type use ===

undefined == null

If one operands is string another is number, convert string to number

If one is boolean and another is non-boolean, convert boolean to number and then perform comparison

While comparing a string or number to an object, try to convert the object to a primitive type and then try to compare

Be careful while comparing objects, identifiers must reference the same objects or same array.

var a = {a: 1};

var b = {a: 1};

a == b //false

a === b //false

var c = a;

a == c//true

a === c //true

**Special note**: NaN, null and undefined will never === another type. NaN does not even === itself.

JavaScript does not care that they are identical and of the same object type.  
**When comparing complex objects, they are equal only when they reference the same object** (i.e., have the same address). Two variables containing identical objects are not equal to each other since they do not actually point at the same object.

## How do you check if a variable is an object?

You can use typeof to determine if variable is an object, however bear in mind that null is actually an object! However null object is 'falsy' thus the following will work:

if(bar && typeof bar === "object") {

console.log('bar is object and is not null');

}

## Closure

 A closure is the combination of a function and the lexical environment within which that function was declared.

A closure is an inner function that has access to the variables in the outer (enclosing) function’s scope chain. The closure has access to variables in three scopes; specifically: (1) variable in its own scope, (2) variables in the enclosing function’s scope, and (3) global variables.

Here is an example:

var globalVar = "xyz";

(function outerFunc(outerArg) {

var outerVar = 'a';

(function innerFunc(innerArg) {

var innerVar = 'b';

console.log(

"outerArg = " + outerArg + "\n" +

"innerArg = " + innerArg + "\n" +

"outerVar = " + outerVar + "\n" +

"innerVar = " + innerVar + "\n" +

"globalVar = " + globalVar);

})(456);

})(123);

In the above example, variables from innerFunc, outerFunc, and the global namespace are all in scope in the innerFunc. The above code will therefore produce the following output:

outerArg = 123

innerArg = 456

outerVar = a

innerVar = b

globalVar = xyz

## What is bind?

bind allows you to borrow a method and set the value of this without calling the function. It simply returns an exact copy of the function with new value of this. You can reuse the same function with new value of this without harming the old one.

var monica = {

name: 'Monica Geller',

total: 400,

deductMontlyFee: function(fee){

this.total = this.total - fee;

return this.name + ' remaining balance is '+ this.total;

}

}

var rachel = {name: 'Rachel Green', total: 1500};

var rachelFeeDeductor = monica.deductMonthlyFee.bind(rachel, 200);

rachelFeeDeductor(); //"Rachel Green remaining balance is 1300"

rachelFeeDeductor(); //"Rachel Green remaining balance is 1100"

## I\*\*\*\*Scope and hoisting: What will you see in the console for the following example?

var a = 1;

function b() {

a = 10;

return;

function a() {}

}

b();

console.log(a);

Answer: 1

Explanation:

function declaration function a(){} is hoisted first and it behaves like var a = function () {};. Hence in local scope variable **a** is created.

If you have two variables with same name (one in global another in local), local variable always get precedence over global variable.

When you set a = 10;, you are setting the local variable a , not the global one. Hence, the value of global variable remains same and you get 1 in the log. ref: js hoisting/scope

Extra: If you didn’t have a function named as "a", you will see 10 in the log.

## I\*\*\*\*How to set a default parameter value for a JavaScript function?

Here email is parameter in which we have set the default value i.e email@domain.com

function function1(name, email)

{

email = typeof email !== 'undefined' ? email : 'defaultemail@domain.com';

console.log('name='+name+', Email= '+email);

}

function1('john','myname@gmail.com');

function1('john');

## How to convert JSON Object to String?

var myobject=['Web','Technology','Experts','Notes']

JSON.stringify(myobject);

## How to convert JSON String to Object?

var jsonData = '{"name":"web technology","year":2015}';

var myobject = JSON.parse(jsonData);

console.log(myobject);

## How do I declare a namespace in JavaScript?

var myNamespace = {

function1: function() { },

function2: function() { }

function3: function() { }

};

myNamespace.function3();

## I\*\*\*\*How do you clone an object?

Answer:

var obj = {a: 1 ,b: 2}

var objclone = Object.assign({},obj);

Now the value of objclone is {a: 1 ,b: 2} but points to a different object than obj.

Note the potential pitfall, though: Object.clone() will just do a shallow copy, not a deep copy. This means that nested objects aren’t copied. They still refer to the same nested objects as the original:

let obj = {

a: 1,

b: 2,

c: {

age: 30

}

};

var objclone = Object.assign({},obj);

console.log('objclone: ', objclone);

obj.c.age = 45;

console.log('After Change - obj: ', obj); // 45 - This also changes

console.log('After Change - objclone: ', objclone); // 45

## Other Q & A: ------------------------------------------------------------

## What is the value of typeof undefined == typeof NULL?

**Answer**: The expression will be evaluated to true, since NULL will be treated as any other undefined variable.

Note: JavaScript is case-sensitive and here we are using NULL instead of null.

## Consider the following code. What will the output be, and why?

(function () {

try {

throw new Error();

} catch (x) {

var x = 1, y = 2;

console.log(x);

}

console.log(x);

console.log(y);

})();

**Answer:**

1

undefined

2

var statements are hoisted (without their value initialization) to the top of the global or function scope it belongs to, even when it’s inside a with or catch block. However, the error’s identifier is only visible inside the catch block. It is equivalent to:

(function () {

var x, y; // outer and hoisted

try {

throw new Error();

} catch (x /\* inner \*/) {

x = 1; // inner x, not the outer one

y = 2; // there is only one y, which is in the outer scope

console.log(x /\* inner \*/);

}

console.log(x);

console.log(y);

})();

## What will be the output of this code?

var x = 21;

var girl = function () {

console.log(x);

var x = 20;

};

girl ();

**Answer:**

Neither 21, nor 20, the result is undefined

It’s because JavaScript initialization is not hoisted.

(Why doesn’t it show the global value of 21? The reason is that when the function is executed, it checks that there’s a local x variable present but doesn’t yet declare it, so it won’t look for global one.)

## What will be the output of the following code:

for (var i = 0; i < 5; i++) {

setTimeout(function() { console.log(i); }, i \* 1000 );

}

Explain your answer. How could the use of closures help here?

**Answer:**

The code sample shown will not display the values 0, 1, 2, 3, and 4 as might be expected; rather, it will display 5, 5, 5, 5, and 5.

The reason for this is that each function executed within the loop will be executed after the entire loop has completed and all will therefore reference the last value stored in i, which was 5.

Closures can be used to prevent this problem by creating a unique scope for each iteration, storing each unique value of the variable within its scope, as follows:

for (var i = 0; i < 5; i++) {

(function(x) {

setTimeout(function() { console.log(x); }, x \* 1000 );

})(i);

}

This will produce the presumably desired result of logging 0, 1, 2, 3, and 4 to the console.

In an ES2015 context, you can simply use let instead of var in the original code:

for (let i = 0; i < 5; i++) {

setTimeout(function() { console.log(i); }, i \* 1000 );

}

## What will this code print?

for (let i = 0; i < 5; i++) {

setTimeout(function() { console.log(i); }, i \* 1000 );

}

**Answer**: It will print 0 1 2 3 4, because we use let instead of var here. The variable i is only seen in the for loop’s block scope.

## What do the following lines output, and why?

console.log(1 < 2 < 3);

console.log(3 > 2 > 1);

**Answer:**

The first statement returns true which is as expected.

The second returns false because of how the engine works regarding operator associativity for < and >. It compares left to right, so 3 > 2 > 1 JavaScript translates to true > 1. true has value 1, so it then compares 1 > 1, which is false.

## How do you add an element at the beginning of an array? How do you add one at the end?

**Answer**:

var myArray = ['a', 'b', 'c', 'd'];

myArray.***push***('end');

myArray.***unshift***('start');

console.log(myArray); // ["start", "a", "b", "c", "d", "end"]

**With ES6, one can use the *spread operator*:**

myArray = ['start', ...myArray];

myArray = [...myArray, 'end'];

Or, in short:

myArray = ['start', ...myArray, 'end'];

## Imagine you have this code:

var a = [1, 2, 3];

a) Will this result in a crash?

a[10] = 99;

b) What will this output?

console.log(a[6]);

**Answer**:

a) It will not crash. The JavaScript engine will make array slots 3 through 9 be “empty slots.”

b) Here, a[6] will output undefined, but the slot still remains empty rather than filled with undefined. This may be an important nuance in some cases. For example, when using map(), empty slots will remain empty in map()’s output, but undefined slots will be remapped using the function passed to it:

var b = [undefined];

b[2] = 1;

console.log(b); // (3) [undefined, empty × 1, 1]

console.log(b.map(e => 7)); // (3) [7, empty × 1, 7]

## What will the following code output and why?

var b = 1;

function outer(){

var b = 2

function inner(){

b++;

var b = 3;

console.log(b)

}

inner();

}

outer();

**Answer:** Output to the console will be “3”.

There are three closures in the example, each with it’s own var b declaration. When a variable is invoked closures will be checked in order from local to global until an instance is found. Since the inner closure has a b variable of its own, that is what will be output.

Furthermore, due to hoisting the code in inner will be interpreted as follows:

function inner () {

var b; // b is undefined

b++; // b is NaN

b = 3; // b is 3

console.log(b); // output "3"

}

**Question: Is 'false' is false?**

Answer: No. Because, it's a string with length greater than 0. Only empty string is false.

**Question: Is ' ' is false?**

Answer: No. Because, it's not an empty string. There is a white space in it.

**Question: What about {}?**

Answer: true. It's an object. An object without any property is an object can't be falsy.

**Question: Tell me about []?**

Answer: This is also truth. It's an array object (array is child of object) is truthy.

**Question: You talked about '' to be falsy. What about new String('')?**

Answer: Though you are passing empty string to the string constructor, it is creating an String object. More precisely a instance of String object. It becomes an object. Hence, it is not false. so, it is truthy.

**Question: Tell me about new Boolean(false)**

Answer: truthy. As it creates an instance of the Boolean object which is an object. Object is truthy.

**Question: Boolean(function(){})**

Answer: true if you pass a truthy value to Boolean, it will be true.

**Question: Boolean(/foo/)**

Answer: true

**Question: true%1**

Answer: 0. When you are trying to find reminder of true, true becomes 1 and reminder of 1 while dividing by 1 is 0. you will get same result if you doe false%1

**Question: ''%1**

Answer: 0

## Rapid Fire: -------------------------------------------------------

**Question: What is typeof []**

Answer: Object. Actually Array is derived from Object. If you want to check array use Array.isArray(arr)

**Question: What is typeof arguments**

Answer: Object. arguments are array like but not array. it has length, can access by index but can't push pop, etc.

**Question: What is 2+true**

Answer: 3. The plus operator between a number and a boolean or two boolean will convert boolean to number. Hence, true converts to 1 and you get result of 2+1

**Question: What is '6'+9**

Answer: 69. If one of the operands of the plus (+) operator is string it will convert other number or boolean to string and perform a concatenation. For the same reason, "2"+true will return "2true"

**Question: What is the value of 4+3+2+"1"**

Answer: 91 . The addition starts from the left, 4+3 results 7 and 7+2 is 9. So far, the plus operator is performing addition as both the operands are number. After that 9 + "1" where one of the operands is string and plus operator will perform concatenation.

**Question: What is the value of "1"+2+4**

Answer: "124". For this one "1" + 2 will produce "12" and "12"+4 will generates "124".

**Question: What is the value of -'34'+10**

Answer: -24. minus(-) in front of a string is an unary operator that will convert the string to a number and will make it negative. Hence, -'34' becomes, -34 and then plus (+) will perform simple addition as both the operands are number.

**Question: What is the value of +'dude'**

Answer: NaN. The plus (+) operator in front of a string is an unary operator that will try to convert the string to number. Here, JavaScript will fail to convert the "dude" to a number and will produce NaN.

**Question: If you have var y = 1, x = y = typeof x; What is the value of x?**

Answer: "undefined"

**Question: for var a = (2, 3, 5); what is the value of a?**

Answer: 5. The comma operator evaluates each of its operands (from left to right) and returns the value of the last operand. ref: MDN

**Question: for var a = (1, 5 - 1) \* 2 what is the value of a?**

Answer: 8

**Question: What is the value of !'bang'**

Answer: false. ! is NOT. If you put ! in front of truthy values, it will return false. Using !! (double bang) is a tricky way to check anything truthy or falsy by avoiding implicit type conversion of == comparison.

**Question: What is the value of parseFloat('12.3.4')**

Answer: 12.3

**Question: What is the value of Math.max([2,3,4,5]);**

Answer: NaN

**Question: 3 instanceof Number**

Answer: false

**Question: null == undefined**

Answer: true

**Question: What is the value of !!function(){};**

Answer: true

**Question: What is the value of typeof bar**

Answer: "undefined"

**Question: What is the value of typeof null**

Answer: "object"

**Question: If var a = 2, b =3 what would be value of a && b**

Answer: 3

**Question: What would be consoled var foo = 'outside'; function logIt(){console.log(foo); var foo = 'inside';} logIt();**

Answer: undefined

**Question: What is -5%2**

Answer:-1. the result of remainder always get the symbol of first operand

**Question: Why .1+.2 != .3**

Answer:

**Question: 42..toString()**

Anwser: "42"

**Question: 4.2..toString**

Anwser: //SyntaxError: Unexpected token .

**Question: 42 . toString()**

Anwser: "42"

**Question: typeof(NaN)**

Anwser:"number"

**Question: 2 in [1,2]**

Anwser: false. Because "in" returns whether a particular property/index available in the Object. In this case object has index 0 and 1 but don't have 2. Hence you get false.

# Type Script

## What are Typescript Types? In Detail?

## What is ECMAScript ES5/ES6?

The ECMAScript is a scripting language which is developed by Ecma International Org.

Currently ECMAScript available in multiple versions that are ES5 and ES6 and both of versions fully supported to Chrome, Firefox, Opera, Safari, and IE etc.

## What is Typescript?

**Typescript is a superset of JavaScript. It is a strongly typed, object oriented and compiled language and this language developed and maintained by Microsoft**. It was designed by “Anders Hejlsberg” at Microsoft.

Typescript also has some additional features like static typing and class-based object-oriented programming, automatic assignment of constructor parameters and assigned null values and so on.

The entire JavaScript program is valid for TypeScript because the entire TypeScript (.ts) file converted to JavaScript (.js) file after compiled source compiled and this process is automatic.

## Why should I use Typescript?

* Supports Object Oriented Programming.
* Typescript adds static typing to JavaScript. Having static typing makes easier to develop and maintain complex apps.
* Angular uses Typescript a lot to simplify relations between various components and how the framework is built in general.
* Provide an optional type system for JavaScript.
* Provide planned features from future JavaScript editions to current JavaScript engines.
* Supports type definitions.

## What's the difference between a TypeScript class and an interface?

**Interface’s job is define “shape” of objects**, helps us keep our programs error-free by providing information about the shape of the data we work with. Interface is powered by some concepts to teach type checker how to verify object with the expected properties is being used:

- Readonly properties → property can be read, can’t be change value

- Index signature →

- Function Types →describe function type

- Class Types: →describe properties and methods for classes

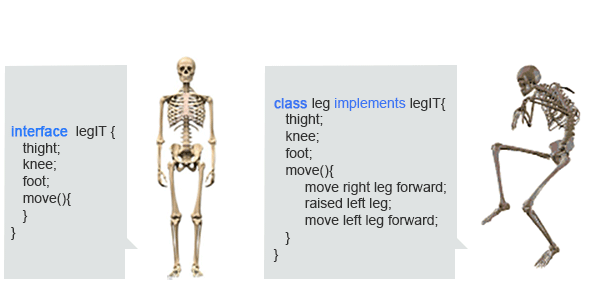
- And some other..

The only job of an interface in TypeScript is to describe a type. While class and function deal with implementation.

Class is built upon concept of low-level language: Inheritance, abstract class, accessors, modifier and some others.

Bear in mind that:

Class and function deal with implementation



https://medium.com/@khiem.office/typescript-differences-between-class-and-interface-b313f0f3dbab

## What are Types in TypeScript?

Typings is the simple way to manage and install TypeScript definitions.

The Type represents the different types of values which are using in the programming languages and it checks the validity of the supplied values before they are manipulated by your programs.

In TypeScript, we define a variable with a “type” and appending the “variable name” with “colon” followed by the type name i.e.

**Number**: the “number” is a primitive number type in TypeScript. There is no different type for float or double in TypeScript.

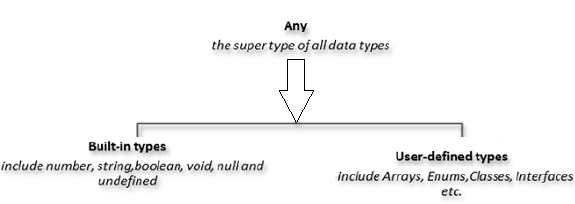
**Boolean**: The “boolean” type represents true or false condition.

**String**: The “string” represents sequence of characters similar to C#.

**Null**: The “null” is a special type which assigns null value to a variable.

**Undefined**: The “undefined” is also a special type and can be assigned to any variable.

**Any** : this data type is the super type of all types in TypeScript. It is also known as dynamic type and using “any” type is equivalent to opting out of type checking for a variable.



## Why type definition (.d.ts) with Typescript in Angular?

A TypeScript definition file contains the type information for written in JavaScript code and the “JavaScript” does not contain type information itself, so “TypeScript” cannot retrieve that information. To solve this problem, we will use the type definition with TypeScript.

## TypeScript Advantages - Pros and Cons!

1. It is purely object-oriented programming.
2. It is support static type-checking.
3. It can be used for client-side and server-side development.
4. Build-in Support for JavaScript Packaging
5. It offers a “compiler” that can convert to JavaScript-equivalent code.
6. It has an API for DOM manipulation.
7. It has a namespace concept by defining a “Module”.
8. Superset of JavaScript
9. ES6 features support

## What is an Interface in TypeScript?

An interface is a way to define a contract on a function with respect to the arguments.

Following code defines an interface and a function that takes a parameter that adheres to that interface.

//USER INTERFACE

interface User { name: string; age: number; address: string}

//FUNCTION USING USER INTERFACE

let userInfo = function(user: User) {

let info = "Hello, " + user.name + " Your Age is - " + user.age + " and Address is -" + user.address;

return info;

}

//USER INFO JSON OBJECT

let info = { name: "Anil", age: 30, address: "Noida, India."};

//RESULT

console.log(userInfo(info));

## What is Functions in TypeScript? How many types you defined in TypeScript?

TypeScript functions are almost similar to JavaScript functions but there are different ways of writing functions in TypeScript.

**Example - Normal function**

function printHello() { console.log('Hello Anil!');}

printHello();

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

**Examples - Anonymous function**

var anonymousFunc = function (num1: number, num2: number): number { return num1 + num2;}

//RESULT

console.log(anonymousFunc(10, 20)); //Return is 30

//RESULT

console.log(anonymousFunc(10, "xyz"));

// error: Argument of type 'number' is not assignable to parameter of type 'string'.

//because return type is number for anonymous function).

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

**Named Function -**

The named function is very similar to the JavaScript function and only one difference - we must declare the type on the passed parameters.

Example –

function addTwoNumer(num1: number, num2: number): number { return num1 + num2;}

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

**Lambda Function/Arrow Function -**

The arrow function is additional feature in typescript and it is also known as a lambda function.

A lambda function is a function without a name.

var addNum = (n1: number, n2: number) => n1 + n2;

In the above, the “=>” is a lambda operator and (n1 + n2) is the body of the function and (n1: number, n2: number) are inline parameters.

For example –

let addNum = (n1: number, n2: number): number => { return n1 + n2; }

let multiNum = (n1: number, n2: number): number => { return n1 \* n2; }

let dividNum = (n1: number, n2: number): number => { return n1 / n2; }

addNum(10, 2);// Result - 12

multiNum(10, 2);// Result - 20

multiNum(10, 2);// Result - 5

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

**Optional Parameters Function -**

We can specify optional properties on interfaces and the property may be present or missing in the object.

In the below example, the address property is optional on the following “User” interface.

For Example as,

//USER INTERFACE

interface User {

name: string;

age: number;

address?: string //Optional

}

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

//FUNCTION USING USER INTERFACE

let userInfo = function(user: User) {

let info = "Hello, " + user.name + " Your Age is - " + user.age + " and Address is -" + user.address;

return info;

}

//USER INFO JSON OBJECT

let info = {

name: "Anil",

age: 30

};

//RESULT

console.log(userInfo(info));

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

**Rest Parameters –**

The Rest parameters do not restrict the number of values that we can pass to a function and the passed values must be the same type otherwise throw the error.

For Example as,

//Rest Parameters

let addNumbers = function(...nums: number[]) {

let p;

let sum: number = 0;

for (p = 0; p < nums.length; p++) {

sum = sum + nums[p];

}

return sum;

}

//The Result

addNumbers(1, 2);

addNumbers(1, 2, 3);

addNumbers(1, 12, 10, 18, 17);

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

**Default Parameters -**

Function parameters can also be assigned values by default.

A parameter can't be declared as optional and default both at the same time.

For Example as,

let discount = function (price: number, rate: number = 0.40) {

return price \* rate;

}

//CALCULATE DISCOUNT

discount(500); // Result - 200

//CALCULATE DISCOUNT

discount(500, 0.45); // Result - 225

## How do you define optional fields in a TypeScript class?

We can specify optional properties on interfaces and the property may be present or missing in the object.

In the below example, the address property is optional on the following “User” interface.

For Example as,

//USER INTERFACE

interface User {

name: string;

age: number;

address?: string //Optional

}

## How Work Automatic Assignment of Constructor Parameters in TypeScript?

Declaring a class with constructor arguments in TypeScript –that is called automatic parameter assignment as,

export class Customer {

constructor(private name: string, age: number, private adrress: string) { }

}

Public, Private, and Protected modifiers as

1. **Public** - accessible outside of the class

2. **Private** - only accessible in the class only

3. **Protected** - accessible in the class and the derived classes

Public modifier by default - When you are not put a modifier (public, private or protected) on your member definition then TypeScript will choose the public by default.

## Which access modifiers are implied when not specified?

Everything in a class is public if not specified. Everything in a module is private unless export keyword is used.

## What is the purpose of the public access modifier for classes in Typescript?

Your sample code means exactly the same in TypeScript. When you don't put a modifier public, private or protected on your member definition then TypeScript will choose the default one which is public.

# ReactiveX (RxJs)

# Angular

## Important Definitions: ----------------------------------------------------

## Components

**Angular components are simply classes that control views and also communicate with other components and services to bring functionality to your app. It has import statements, Decorator and export classes.**

## Directives

***Directives are*** angular decorator ***decorated on the HTML elements, specific to your application. They are essentially functions that execute when the Angular compiler finds them in the DOM.***

* ***Directives are instructions which tell Angular to do something.***
* ***Directives allow you to attach behaviour to element in the DOM.***

**Components Directives**

**Structural Directives**

**Attribute Directives**

## Services

**An Angular service is a class along with its associated properties and Methods that encapsulates some sort of functionality that can be included into Angular components via dependency injection.**

## AOT

**AOT used to compiles the angular components and templates to native JavaScript and HTML during the build time instead of run-time.**

## Lazy loading

**Lazy loading enables us to load only the module user is interacting and keep the rest to be loaded at run-time on demand.**

## Promise

**Promises are chainable concurrency primitive that acts as placeholder for a future value.**

new Promise((resolve, reject) => {

setTimeout(() => {

resolve('result')

}, 100)

})

.then(console.log)

.catch(console.error)

## Observables

**Observables open up a continuous channel of communication in which multiple values of data can be emitted over time. From this we get a pattern of dealing with data by using array-like operations to parse, modify and maintain data.**

## Route Guards

**Route guards are interfaces which can tell the router whether or not it should allow navigation to a requested route.**

**CanActivate**

**CanActivateChild**

**CanDeactivate**

**CanLoad**

**Resolve**

## Pipes

**A pipe is a filter that takes in data as input and transforms it to a desired output.**

## Imp Websites Q&A:------------------------------------------------------

[**https://www.code-sample.com/2017/04/angular-4-interview-questions-and.html**](https://www.code-sample.com/2017/04/angular-4-interview-questions-and.html)

[**https://www.wisdomjobs.com/e-university/angular-2-interview-questions.html**](https://www.wisdomjobs.com/e-university/angular-2-interview-questions.html)

[**https://github.com/Yonet/Angular-Interview-Questions**](https://github.com/Yonet/Angular-Interview-Questions)

[**https://github.com/khan4019/angular-interview-questions**](https://github.com/khan4019/angular-interview-questions)

[**https://www.codeproject.com/Articles/1169073/Angular-Interview-Questions**](https://www.codeproject.com/Articles/1169073/Angular-Interview-Questions)

## Q & A :----------------------------------------------------------------------

## What is Angular 2+?

Angular is a Typescript + javascript +Dart framework used for creating web and mobile applications. It allows you to use HTML as your template language and enables you to extend HTML’s syntax to express your application’s components clearly.

1. Smaller & Faster Apps
2. View Engine Size Reduce
3. Animation Package
4. NgIf and ngFor Improvement
5. Template
6. NgIf with Else
7. Use of AS keyword
8. Pipes
9. HTTP Request Simplified
10. Apps Testing Simplified
11. Introduce Meta Tags
12. Added some Forms Validators Attributes
13. Added Compare Select Options
14. Enhancement in Router
15. Added Optional Parameter
16. Improvement Internationalization

## What are the differences between angular 2 and angular 4 version?

## I\*\*\*\*What is angular CLI?

## I\*\*\*\*What are the fundamentals concepts of Angular?

* Components
* Dependency injection, and
* Bindings

## I\*\*\*\*What are advantages of Angular?

* Angular has better performance.
* Angular has more powerful template system.
* Angular provide simpler APIs, lazy loading and easier to application debugging.
* Angular is much more testable.
* Angular provides to nested level components.
* The Angular execute run more than two programs at the same time.

The Angular architecture identifies the eight main building blocks as.

1. Module

2. Component

3. Template

4. Outpouts

5. Data Binding

6. Directive

7. Service

8. Dependency Injection

## I\*\*\*\*What is AOT (Ahead-of-Time) Compilation?

**AOT used to compiles the angular components and templates to native JavaScript and HTML during the build time instead of run-time.**

The compiled HTML and JavaScript are deployed to the web server so that the compilation and render time can be saved by the browser. It is the big advantage to improve the performance of applications.

Advantages of AOT

Faster download

Faster Rendering

Lesser Http Requests

Detect error at build time

Disadvantages of AOT

AOT only works only with HTML and CSS and not for other file

We need to maintain AOT version of bootstrap file.

We need to clean-up step before compiling.

## I\*\*\*\*How would you optimize the Angular Application for Better Performance?

Consider AOT compilation.

Consider lazy loading

Remove un-necessary import statements.

Remove 3rd party library from your application, if not required.

Remove your application dependencies if not required.

Well, optimization depends on the type and size of application and many other factors. But in general, I would consider the following points while optimizing the angular app:

Consider AOT compilation.

Make sure the application is bundled, uglified, and tree shaking is done.

Make sure the application doesn’t have un-necessary import statements.

Make sure that any 3rd party library, which is not used, is removed from the application.

Have all dependencies and dev-dependencies are clearly separated.

I would consider lazy loading instead of fully bundled app if the app size is more.

## I\*\*\*\*What are differences between Constructors and OnInit?

Could anyone tell me about the usage of ngOnInit if we already have a constructor?

Constructors:-

* The constructor is a default method runs when component is being constructed.
* The constructor is a typescript feature and it is used only for a class instantiations and nothing to do with Angular.
* The constructor called first time before the ngOnInit().

ngOnInit:-

* The ngOnInit event is an Angular life-cycle event
* ngOnInit is called after the constructor and ngOnInit is called after the first ngOnChanges.
* The ngOnChanges is called when a @input or @output binding value changes otherwise the constructor is OK.

## I\*\*\*\*What is the order of life-cycle hooks?

Angular Complete lifecycle hook interface inventory:-

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

Angular Lifecycle Events Log:-

1. onChanges
2. onInit
3. doCheck
4. afterContentInit
5. afterContentChecked
6. afterViewInit
7. afterViewChecked
8. doCheck
9. afterContentChecked
10. afterViewChecked
11. onChanges
12. doCheck
13. afterContentChecked
14. afterViewChecked
15. onDestroy

## I\*\*\*\*What is Components in Angular?

* Angular Components are meta-data annotation that is used to register the components.
* Components are used to create UI widgets.
* Components are used to split to application into smaller parts.
* Only one component is used per DOM element.
* In Angular components selector and template or templateUrl are mandatory in the components.

**Angular components are simply classes that control views and also communicate with other components and services to bring functionality to your app. It has import statements, Decorator and export classes.**

## I\*\*\*\*What are differences between Components and Directives?

|  |  |
| --- | --- |
| [**@Components**](http://www.code-sample.com/2016/04/angular-2-components-vs-directives.html) | [**@Directive**](http://www.code-sample.com/2016/04/angular-2-components-vs-directives.html) |
| 1.       @Component meta-data annotation is used to register the components. | @Directive meta-data annotation is used to register the directives. |
| 2.       The components are used to create UI widgets. | The directives are used to add behavior to existing DOM elements. |
| 3.       The components are used to split to application into smaller parts. | The directives are used to design a reusable component. |
| 4.       Only one component is used per DOM element. | More than one directive is used per DOM element. |
| 5.       In the components, @View, template and templateUrl are mandatory in the components. | The directive does not have @View etc. |

A component is a directive with a template and the @Component decorator is actually a @Directive decorator extended with template-oriented features.

## I\*\*\*\*What is @Inputs and @Outputs in Angular?

@Input allows you to pass data into your controller and templates through html and defining custom properties.

@Input is used to define an input for a component; we use the @Input decorator.

@Output decorator is used to binds a property of a component to send the data from child component to parent component and this is a one-way communication.

@Output decorates output properties and its binds a property of the type of angular EventEmitter.

## I\*\*\*\*How to components communicate with each other?

In Angular a component can share data and information with another component by passing data or events. A component can be used inside another component, thus creating a component hierarchy. The component being used inside another component is known as the child component and the enclosing component is known as the parent component. Components can communicate to each other in various ways, including:

Using @Input()

Using @Output()

Using Services

Parent component calling ‘ViewChild’

Parent interacting with child using a local variable

## I\*\*\*\*What is an Angular Service?

An Angular service is a class along with its associated properties and Methods that encapsulates some sort of functionality that can be included into Angular components via dependency injection.

Following three steps are required to create an Angular Service.

1. Import the injectable member.
2. Add the @Injectable Decorator.
3. Export Service class
4. **import** { Injectable } from '@angular/core';
6. @Injectable()
7. **export** **class** MyService {
9. }

Features:

1. Services are singleton objects.
2. Services are capable of returning the data in the form promises or observables.
3. Service class is decorated with Injectable decorator.
4. The Injectable decorator is required only if our service class is making use of some Angular injectable like Http, Response and HttpModule service within it.

## I\*\*\*\*What is Dependency Injection (DI) in Angular?

**Dependency injection is a process where we inject the dependent objects rather than consumer creating the objects.**

Angular Dependency Injection consists of three things.

**Injector** :- The injector object use to create instances of dependencies.

**Provider** :- A provider is help to injector for create an instance of a dependency. A provider takes a token and maps that to a factory function that creates an object.

**Dependency** :- A dependency is the type of which an object should be created.

## I\*\*\*\*What is @Injectable()? Why Use?

@Injectable () is a decorator.

If we add the decorator, the metadata design:paramtypes is added to the compiled .js, and the dependency injection can do its job. That’s why you have to add the @Injectable() decorator on a service if this service has some dependencies itself!

@Injectable () marks a class as available to an injector for instantiation. An injector reports an error when trying to instantiate a class that is not marked as @Injectable ().

Example:-

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

import { Http, Response } from '@angular/http';

@Injectable()

export class UserService {

constructor(private \_http: Http) {

}

getAPIUsers(apiUrl) {

return this.\_http.get(apiUrl).map((data: Response) => data.json());

}

}

<http://blog.ninja-squad.com/2016/12/08/angular-injectable/>

## What is @Inject()? Why Use?

@Inject() is a special technique for letting Angular know that a parameter must be injected.

How to use Dependency Injection (DI) correctly in Angular?

The basics Steps of Dependency injection,

1. A class with @Injectable () to tell Angular2 that it’s to be injected “UserService”.

2. A class with a constructor that accepts a type to be injected.

Example, UserService marked as @Injectable as,

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

import { Http, Response } from '@angular/http';

import 'rxjs/add/operator/map';

@Injectable()

export class UserService {

constructor(private \_http: Http) {

}

getAPIUsers(apiUrl) {

return this.\_http.get(apiUrl).map((data: Response) => data.json());

}

getAppUsers(apiUrl) {

return this.\_http.get(apiUrl).map((data: Response) => data);

}

}

## I\*\*\*\*Why are components not marked with @injectable annotation, but services need to be?

What’s the difference between for..in and for..of?

## I\*\*\*\*What are the differences between Observables & Promises?

Promise: - Promises are only called once and it can return only a single value at a time and the Promises are not cancellable. A Promise handles a single event when an async operation completes or fails.

Observables: - Observables handle multiple values over time and it can return multiple values and the Observables are cancellable.

The Observables are more advanced than Promises.

Promises vs Observables

**Promises**:

Returns a single value

Not cancellable

**Observables**:

Works with multiple values over time

Cancellable

Supports map, filter, reduce and similar operators

Proposed feature for ES 2016

Use Reactive Extensions (RxJS)

An array whose items arrive asynchronously over time

## I\*\*\*\*What is Pipes? Why use Pipes?

**A pipe is a filter that takes in data as input and transforms it to a desired output.**

There are two types of pipes i.e.

* Pure
* Impure

Pipe class implements the “PipeTransform” interfaces transform method that accepts an input value and returns the transformed result.

There will be one additional argument to the transform method for each parameter passed to the pipe.

The “@Pipe” decorator allows us to define the pipe name that is globally available for use in any template in the across application.

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

**name: 'barcode',**

    pure: false

})

export class BarCodePipe implements PipeTransform {

    transform(value: string, args: any[]): string {

        if (!value) {

            return '';

        }

        return "\*\*\*\*-\*\*\*\*\_" + (value.length > 8 ? (value.length - 8): '')

    }

}

## I\*\*\*\*What is Directives in Angular?

***Directives are*** angular decorator ***decorated on the HTML elements, specific to your application. They are essentially functions that execute when the Angular compiler finds them in the DOM.***

* ***Directives are instructions which tell Angular to do something.***
* ***Directives allow you to attach behaviour to element in the DOM.***

There are 3 types of directives in Angular.

1. **Components Directives** - directives with a template

A component is a directive with a template and the @Component decorator is actually a @Directive decorator extended with template oriented features.

1. **Structural Directives** - change the DOM layout by adding and removing DOM elements.

The Structural directives are responsible for HTML layout and It is using Angular for reshape the DOM's structure and also removing, or manipulating elements.

1. **Attribute Directives** - change the appearance or behaviour of an element, component, or other directive.

<p myHighlight>Highlight me!</p>

import { Directive, ElementRef, Input } from '@angular/core'; @Directive({ selector: '[myHighlight]' })

export class HighlightDirective { constructor(el: ElementRef) { el.nativeElement.style.backgroundColor = 'yellow'; } }

## I\*\*\*What is Template in Angular?

The template consists of HTML, CSS, and Angular directives contained in just one HTML file.

* {{}}: Is use to rendering the HTML elements.
* []: Is use to binding properties.
* (): Is use to handling your events.
* [()]: Is use to data binding.
* \*: Is use to asterisk Operations like \*ngFor="let item of items; let i=index;”

## I\*\*\*\*What is Routing Concepts in Angular?

The Router is use to map applications URLs to application components.

Angular provides 3 different components for routing configuration:

1. **Routes** is the configuration to describe application’s different routes
2. **RouterOutlet** is a “placeholder” component that holds the view for each route
3. **RouterLink** is a directive to link to routes

Routing is a mechanism which enables user to navigate between views/components. Angular simplifies the routing and provide flexibility to configure and define at module level (Lazy loading).

Routes: - The Routes is uses to describe our application's Routes. The “RouterModule.forRoot” method in the module imports to configure the router.

Five concepts that need Routes Representation

1. Path (a part of the URL)

2. Route Parameters

3. Query/Matrix Parameters

4. Name outlets

5. A tree of route segments targeting outlets

Syntax:-

RouterModule.forRoot([

{ path: '', redirectTo: 'home', pathMatch: 'full' },

{ path: 'home/:id', component: HomeComponent }, //HERE ID IS A ROUTE PARAMETER.

{ path: 'login', component: LoginComponent },

{ path: 'registration', component: RegistrationComponent },

{ path: '\*\*', redirectTo: 'home' }

])

Router Imports - The Angular Router is an optional service that presents a particular component view for a given URL i.e.

import { RouterModule, Routes } from '@angular/router';

Router-outlet directive: - Router-outlet directive is used to render the components for specific location of your applications. Both the template and templateUrl render the components where you use this directive.

Syntax :- <router-outlet></router-outlet>

The Route Params: - The route parameter is used to map given URL's parameters based on the rout URLs and it is an optional parameters for that route.

Syntax: - params: {[key: string]: string}

Router-link directive: - Router-link directive is used to link a specific part of your applications.

Syntax :- <router-link></router-link>

## I\*\*\*\*What is Router Directives?

Angular Router Module has 3 different directives. Such as:

1. router-outlet
2. router-link
3. routerLinkActive

**router-outlet:**

router-outlet is a component from angular/router library. The router is the placeholder to display views inside <router-outlet> tags. As the routes changes, the view inside the <router-outlet> tags also change accordingly.

<div class="container-fluid">

<router-outlet></router-outlet>

</div>

**router-link:**

router-link directive is an alternative of HTML href property. The syntax is as following:

<a [routerLink]="['/url']">Url Title</a>

router-link directive can also handle query parameter. For example, once we land in /hoem/events page, it shows a list of events , if we want to go to directly one of the events page, we can just put the iid of that specific event and can land of the event details page for event 01. For this purpose, we use [queryParams] binding. [queryParams] takes an object such as {‘id’:01} and can

Firstly, We have bound router-link directive to events array as following:

<a [routerLink]="['/home/events]">Events</a>

Now inside app.router.ts we have configure the queryParams binding as following:

this.router.navigate(['/home/events', event.id]);

Otherwise, if we want to pass the queryParams in an object , we can also do that as following:

<a [routerLink]="['/home/events, { id: 'event01' }]">Event 01 </a>

OR

<a [routerLink]="['/home/events, 1]">Event 1</a>

routerLinkActive:

The RouterLinkActive directive toggles css classes for active RouterLinks based on the current RouterState. This cascades down through each level in our route tree, so parent and child router links can be active at the same time. To override this behavior, we can bind to the [routerLinkActiveOptions] input binding with the { exact: true } expression. By using { exact: true }, a given RouterLink will only be active if its URL is an exact match to the current URL.

<li class="nav-item px-1">

<a class="nav-link" routerLinkActive="active" [routerLink]="['/home/events']"><i class="icon-puzzle"></i>Events</a>

</li>

## I\*\*\*\*What is the purpose of NgModule?

An NgModule is a class adorned with the @NgModule decorator function. @NgModule takes a metadata object that tells Angular how to compile and run module code.

The purpose of introducing @NgModule() was to allow lazy loading with the router.

the @NgModule annotation is what actually defines the module

we can list the components, directives, and pipes that are part of the module in the declarations array

we can import other modules by listing them in the imports array (Imports is only for NgModules, not for component or directives).

we can list the services that are part of the module in the providers array but read further on why this should only be used in some cases

## I\*\*\*\*What is an entry component?

An entry component is any component that Angular loads imperatively by type.

A component loaded declaratively via its selector is not an entry component

Angular automatically adds the following types of components to the module’s entry Components:

The component in the @NgModule.bootstrap list.

Components referenced in router configuration.

You don't have to mention these components explicitly, although doing so is harmless.

**A bootstrapped component is an entry component that Angular loads into the DOM during the bootstrap (application launch) process**. Other entry components are loaded dynamically by other means, such as with the router.

**The @NgModule.bootstrap property tells the compiler that this is an entry component and it should generate code to bootstrap the application with this component.**

## I\*\*\*\*What's the difference between RouterModule.forChild and RouterModule.forRoot?

## I\*\*\*\*What is Lazy Loading and How to enable Lazy Loading?

**Lazy loading enables us to load only the module user is interacting and keep the rest to be loaded at run-time on demand.**

Lazy loading speeds up the application initial load time by splitting the code into multiple bundles and loading them on demand.

* Each Angular application must have one main(root) module that is called “AppModule” and your code should be splitted into various child modules based on your applications.
* We do not require to import or declare lazily loading module in root module.
* Add the route to top level routing and takes routes array and configures the router.
* Import module specific routing in the child module.

## I\*\*\*\*What is Route Guard?

<https://medium.com/@ryanchenkie_40935/angular-authentication-using-route-guards-bf7a4ca13ae3>

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

## I\*\*\*\*Restrict User not to access pages which is no authorized?

## I\*\*\*\*What are cookies in Angular?

Cookies are small packages of information that are typically stored by your browser and websites tend to use cookies for multiple things.

In order to play about with cookies in Angular we’ll have to install the angular2-cookie library by typing the following within our project:

npm install angular2-cookie --save

**Setting Cookies**

In order to set cookies using this library we would have to use the following method:

this.\_cookieService.put('test', 'test');

**Retrieving Cookies**

Once we have successfully set a cookie, we can then retrieve that same cookie by using the .get(cookie) method like so:

this.\_cookieService.get('test');

Angular cookies concept is very similar to the Angular 1.x but Angular added only one extra method to remove all cookies i.e. removeAll()

The All cookie methods are

1. get()

2. getObject()

3. getAll()

4. put()

5. putObject()

6. remove()

7. removeAll() - This is new one in angular

## I\*\*\*\*Can we create two Components with the same name in two different .ts files?

You can't have two root components with the same selector in your page, you also can't have two @Component() decorators on the same class.

If your components have different selectors, just run bootstrap for each root component

@Component({

selector: 'app',

template: '<h1>AppComponent1</h1>'

})

export class AppComponent1 { }

@Component({

selector: 'appTwo',

template: '<h1>AppComponent2</h1>'

})

export class AppComponent2 { }

bootstrap(AppComponent1)

bootstrap(AppComponent2)

## I\*\*\*\*How to handle errors in Angular using error handler module.

## I\*\*\*\*How would you intercept 404 errors in Angular?

**Http sends errors down the error stream of an observable so you will need to catch them with .catch**

return next.handle(request)

.do(event => {

if (event instanceof HttpResponse) {

this.logger.logDebug(event);

}

})

.catch(err => {

console.log('Caught error', err);

return Observable.throw(err);

});

## I\*\*\*\*How to make sure that single instance will be used in an entire application?

Angular maintains a single instance per provider.

**Ensure you provide a service only once and DI will ensure that there is only one single instance in your application.**

If you provide a service on a component @Component ({ ..., providers: [...]}), then there will be as many instances as component instances.

If you provide a service only in providers of AppModule or providers of modules imported to AppModule, then there will only be a single instance for your whole application:

@NgModule({

providers: [...],

imports: [...]

})

export class AppModule {}

## I\*\*\*\*What is a Root Module?

Each application only has one root module, and each component, directive and pipe should only be associated with a single module.

// app.module.ts

@NgModule({

imports: [BrowserModule, FormsModule, HttpModule],

declarations: [

AppComponent,

VehicleListComponent,

VehicleSelectionDirective,

VehicleSortingPipe

],

providers: [

LoggerService,

VehicleService,

UserProfileService

],

bootstrap: [AppComponent],

})

export class AppModule { }

## I\*\*\*\*What are Event Emitters and how it works in Angular?

Any change occurred in the component always gets propagated from the current component to all its children in hierarchy. If the change from one component needs to be reflected to any of its parent component in hierarchy, we can emit the event by using Event Emitter api.

In short, EventEmitter is class defined in @angular/core module which can be used by components and directives to emit custom events.

@output() somethingChanged = new EventEmitter();

We use somethingChanged.emit(value) method to emit the event. This is usually done in setter when the value is being changed in the class.

This event emit can be subscribed by any component of the module by using subscribe method.

myObj.somethingChanged.subscribe(val) => this.myLocalMethod(val));

## I\*\*\*\*How do you use one way and two way data binding?

## I\*\*\*\*Which is better, reactive or template-driven?

**Template Driven Forms Features**

* Easy to use
* Suitable for simple scenarios and fails for complex scenarios
* Similar to angular 1
* Two way data binding(using [(NgModel)] syntax)
* Minimal component code
* Automatic track of the form and its data(handled by Angular)
* Unit testing is another challenge

**Reactive Forms Features**

* More flexible, but needs a lot of practice.
* Handles any complex scenarios.
* No data binding is done(Immutable data model preferred by most developers)
* More component code and less HTML mark-up
* Reactive transformations can be made possible such as
* Handling an event based on a debounce time
* Handling events when the components are distinct until changed
* Adding elements dynamically
* Easier Unit testing

## I\*\*\*\*How do you use a JavaScript (Non TypeScript) third party lib in an Angular App?

Yes, you need the library:

npm install <library> --save

After that you need the type definition:

tsd install <library> --save

and at last, you need to include it in your project:

// Note that the package name might be different

import {<library>} from '<library>';

## I\*\*\*\*ngFor vs. ngForOf

The [ngFor] is not a type safe

The [NgForOf] is a type Safe

The [NgFor] directive instantiates a template once per item from iterate

The [ngFor] and [ngForOf] are actually the selectors of the [NgForOf] directive and it is not two distinct things

The [ngFor] will be works like as collections

The [ngForOf] will be works like as generics

## I\*\*\*\*HOW TO GET AND LOG AN ERROR IN ANGULAR 4?

Two types of errors –

If the backend returns an unsuccessful response like – 404, 500 and so on

If something goes wrong in the client side like -network error etc.

In the both cases – We are using HttpErrorResponse and return the useful information on what went wrong in this call!

## Authentication:------------------------------------------------------------

## HOW ARE JWTS USED TO AUTHENTICATE ANGULAR 4 APPLICATIONS?

In Annular, the following Steps are used to building authentication and authorization for RESTful APIs and applications. It might help you –

The users send their credentials to the server which is verified by the database credentials. If everything is verified successfully, the JWT is sent back to them.

The JWT is saved in the user’s browser in local storage or in a cookie and so on.

The presence of a JWT saved in the browser is used as an indicator that a user is currently logged in.

The expiry time of JWT is continually checked to maintain an authenticated state in the Angular applications.

The client side routes are protected and access by authenticated users only.

When user sends the XHR requests for APIs, the JWT gets sent an Authorization header using your cookies or Bearer.

When XHR requests coming on the server, before send back the responses it’s validated first with configured app’s secret keys. If everything is looking good then returns successfully responses other send the back to the bad request.

There are several open source libraries are available for angular which are helps with JWTs and has the ability to Decode the JWT, Authorization header to XHR requests and so on.

## WHAT IS JSON WEB TOKEN?

JSON Web Token (JWT) is an open standard which used for securely transmitting information between parties as a JSON object.

The JWTs can be signed with –

HMAC algorithm

RSA algorithm

WHEN SHOULD YOU USE JSON WEB TOKENS?

There are some scenarios where we can used JSON Web Tokens –

Authentication

Information Exchange

## WHAT IS THE JSON WEB TOKEN STRUCTURE?

The JSON Web Tokens consist of three parts separated by dots (.), which are:

Header

Payload

Signature

## Few More Q & A:---------------------------------------------------------------

## How would you make use of NgOnInit()?

The **ngOnInit** is invoked when the component is initialized and invoked only once when the directive is instantiated. It is a best practice to implement these life-cycle interfaces.

## What is the Best way to Declare and Access a Global Variable in Angular?

Put the global variables in a file an export them.

//

// ===== File globals.ts

//

'use strict';

export const sep='/';

export const version: string="22.2.2";

To use globals in another file use an import statement:

import \* as myGlobals from './globals';

## Briefly explain Angular Metadata

In Angular, different decorators are used to pass variables to class. Such as

@NgModules decorator is used for Modules.

@Component decorator is used to pass variables to class.

Decorators uses metadata to attach variable to component or module class.

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

FormsModule,

HttpModule

],

providers: [],

exports:[

AppComponent

],

bootstrap: [AppComponent]

})

Here @NgModule pass a metadata, which is kind of json variable with properties such as declaration, imports, providers, export and bootstrap.

Similarly, @Component decorator also used metadata.

@Component({

selector: 'app-root',

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

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

})

In brief, metadata defines the properties for any module or component. Metadata helps any class to process and execute.

## What kind of classes can you import (Meta) in an angular module?

<http://magemello.github.io/articles/003-first-angular-2-app.html>

## What are a few subclasses of @Injectable?

## What types of pipes are supported in Angular?

## Why are Angular modules needed?

An Angular module allows Angular to define a context for compiling templates. For example, when Angular is parsing HTML templates, it's looking for a certain list of components, directives and pipes.

Each HTML tag is compared to that list to see if a component should be applied on top of it, the same goes for each attribute. The problem is: how does Angular know which components, directives and pipes should it be looking for while parsing the HTML?

That is when Angular modules come in, they provide that exact information in a single place.

So in summary, we can say the following about Angular modules:

they are essential for template parsing, both in the Just In Time or Ahead Of Time Compilation scenarios as we will see

they are also very useful simply as documentation for grouping related functionality

They can be used to clarify which components and directives are meant to be used publicly vs internal implementation details, as we will soon see

## What is Traceur compiler in Angular?

The Traceur is a JavaScript compiler. The Traceur compiler used to allow us to use the features from the future. The Traceur compiler is fully supported to ECMAScript (ES6) and ES.vNext also.

The main goal of Traceur compiler is to inform the designs of new JavaScript features and also allow us to write the code in better manners and it also prefer, tell us to use design patterns.

Now the days Traceur compiler are broadly used in Angularv2.0 because Angular v2.0 are fully used to ES5 and ES6

**Traceur compiler compiles ECMAScript Edition 6 (ES6) (including classes, generators and so on) code on the fly to regular Javascript (ECMAScript Edition 5 [ES5]) to make it compatible for the browser.**

**Traceur itself is written in ES6, compiled to ES5**.

## What is the use of codelyzer in angular applications?

All enterprise applications follows a set of coding conventions and guidelines to maintain code in better way. Codelyzer is an open source tool to run and check whether the pre-defined coding guidelines has been followed or not. Codelyzer does only static code analysis for angular and typescript project.

Codelyzer runs on top of tslint and its coding conventions are usually defined in tslint.json file. Codelyzer can be run via angular cli or npm directly. Editors like Visual Studio Code and Atom also supports codelyzer just by doing a basic settings.

To set up the codelyzer in Visual Studio code, we can go to File -> Preferences -> User

Settings and add the path for tslint rules.

Hide Copy Code

{

"tslint.rulesDirectory": "./node\_modules/codelyzer",

"typescript.tsdk": "node\_modules/typescript/lib"

}

To run from cli: ng lint.

To run from npm: npm run lint

## What is lazy loading and How to enable lazy loading in angular?

Lazy lading enables us to load only the module user is interacting and keep the rest to be loaded at runtime on demand.

Every Angular application must have one main module say AppModule. The code should be spitted into various child modules (NgModule) based on the application business case.

## What is module and how to modularize application?

## What is FormGroup and FormControl in angular? Why is it used for?

## What is change in BrowserAnimationsModule from angular version 2 to 4?

## What is angular decoration and why is used for?

## What is interpolation in angular component?

## How to apply animation to the angular component?

## How can you dynamically load the component?

To insert a component dynamically you can use ViewContainerRef.createComponent()

For a declarative approach you can use a helper component like

@Component({

selector: 'dcl-wrapper',

template: `<div #target></div>`

})

export class DclWrapper {

@ViewChild('target', {read: ViewContainerRef}) target;

@Input() type;

cmpRef:ComponentRef;

private isViewInitialized:boolean = false;

constructor(private resolver: ComponentResolver) {}

updateComponent() {

if(!this.isViewInitialized) {

return;

}

if(this.cmpRef) {

this.cmpRef.destroy();

}

this.resolver.resolveComponent(this.type).then((factory:ComponentFactory<any>) => {

this.cmpRef = this.target.createComponent(factory)

});

}

ngOnChanges() {

this.updateComponent();

}

ngAfterViewInit() {

this.isViewInitialized = true;

this.updateComponent();

}

ngOnDestroy() {

if(this.cmpRef) {

this.cmpRef.destroy();

}

}

}

## Why do we need provider aliases? And how do you create one?

## What's the expression context in Angular? Explain it with an example.

## What is the NgForTrackBy directive?

## Can we import a module twice?

That's not a problem. When three modules all import Module 'A', Angular evaluates Module 'A' once, the first time it encounters it, and doesn't do so again.

Angular doesn't like modules with circular references, so don't let Module 'A' import Module 'B', which imports Module 'A'.

## Can you re-export classes and modules?

Absolutely.

Modules are a great way to selectively aggregate classes from other modules and re-export them in a consolidated, convenience module.

A module can re-export entire modules, which effectively re-exports all of their exported classes. Angular's own BrowserModule exports a couple of modules like this:

content\_copy

exports: [CommonModule, ApplicationModule]

A module can export a combination of its own declarations, selected imported classes, and imported modules.

You can export any declarable class—components, directives, and pipes—whether it's declared in this module or in an imported module. You can re-export entire imported modules, which effectively re-exports all of their exported classes. A module can even export a module that it doesn't import.

## How can you connect to remotely deployed backed while in development?

## How do you reference the host of a component?

## UTube

* What are Directives? Explain attribute Directive?
* How many types of directives are there in Angular 2?
* How are events attached to Directive?
* How do we pass external data into Directive?
* What are structural Directives? Name few Angular built-in structural Directives?
  + Structural directive are responsible for HTML layout.
  + They shape or reshape the DOM’s structure, typically by adding, removing or manipulating element.
  + Every structural directive start with \*
* What is @HostListner ?

## Real Interview Q & A:-----------------------------------------------------

1 - Explain the process by which your application is bootstrapped. How does Angular know which component to load?

2 - If I have two components, which have a parent-child relationship, how would I push data to the child component where it will be displayed, from an action in the parent component? How would I PULL data from the child component to the parent? Each of these operations will be triggered from an event in the component doing the pushing, or doing the pulling.

3 - Given two components with no direct parent/child relationship, how can I share data between the components if they are both children of the same parent? What if they don’t have the same parent?

4 - Explain how to inject a service into a component or another service. If each injection, into multiple components for example, should result in a new instance of the service, how would you handle that?

5 - Explain how component routing works in Angular. How would I define routes and sub-routes?

6 - Talk to me about security in Angular applications. How would I restrict access to routes? You may use any back-end authentication and authorization mechanisms you wish (because we don’t care about the back-end for this question set).

7 - How would you transform data using pipes in Angular? (ask for examples of using built-in pipes here) How do you develop your own pipe?

8 - How do you make AJAX calls, and how do you handle the returned data? (this leads to #9)

9 - Explain the purpose of Observables and Promises, and when each should be used (ie. when would you want to use a promise instead of an observable)? Explain how you would subscribe to the observable to update the DOM asynchronously when the observed object changes.

5 - What are the methods that Angular provides for creating form objects? When would you prefer each one? How do you handle data validation in the forms? I want to populate fields using the results of an AJAX call, and conditionally populate (depending on the results of the AJAX call) certain fields when the form loads, how would you do this?

11 - How would you handle a situation in which multiple components may be updating, displaying - or both updating and displaying - the same data? How would you create an architecture to keep the data being displayed synchronized properly across all of the components?

# Nodejs

<https://blog.risingstack.com/node-js-interview-questions-and-answers-2017/>

## What is an error-first callback?

Error-first callbacks are used to pass errors and data as well. You have to pass the error as the first parameter, and it has to be checked to see if something went wrong. Additional arguments are used to pass data.

fs.readFile(filePath, function(err, data) {

if (err) {

// handle the error, the return is important here

// so execution stops here

return console.log(err)

}

// use the data object

console.log(data)

})

## How can you avoid callback hells?

There are lots of ways to solve the issue of callback hells:

modularization: break callbacks into independent functions

use a control flow library, like async

use generators with Promises

use async/await (note that it is only available in the latest v7 release and not in the LTS version - you can read our experimental async/await how-to here)

## When are background/worker processes useful? How can you handle worker tasks?

Worker processes are extremely useful if you'd like to do data processing in the background, like sending out emails or processing images.

There are lots of options for this like RabbitMQ or Kafka.

# Setup development environment for Angular

## Angular application can be developed using:

**angular-cli**: This is a command line tool for Angular application development which support live coding and deployment instantly. This is much easier and much straight forward.

**systemjs**: This is the traditional way of developing angular application. This way has been described in details later in Part-2.

## Development Angular Application Using ANGULAR-CLI:

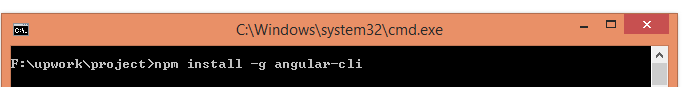
### Pre-requirement:

1. Install latest Node.js

* Download NodeJs 7.0 + version from https://nodejs.org/en/
* Install NodeJs
* Check if Nodejs installation is correct by running the command from command prompt ‘npm -version’

1. Install angular-cli (https://cli.angular.io/)

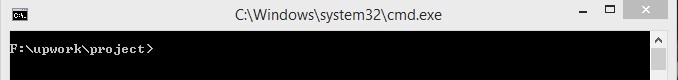
* Install angular-cli by following command



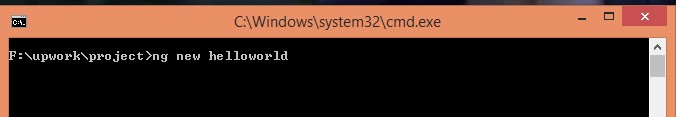
The next step is to create the application from command line.

For example, if I want to create an angularjs application in folder project named ‘helloworld’. The steps are described below.

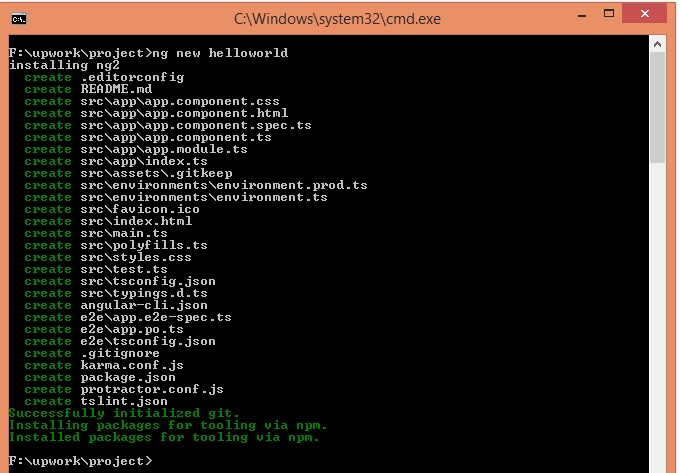
1. Open the command prompt on the workspace folder called ‘project’.



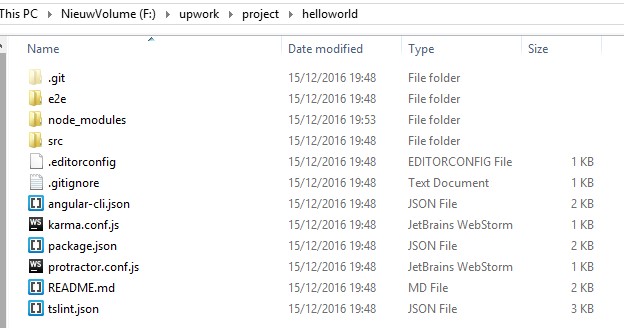
1. Run command ‘ng new <project name>’. Lets suppose the project name is ‘helloworld’. This command will create a new project named ‘helloworld’. This may take few minutes.



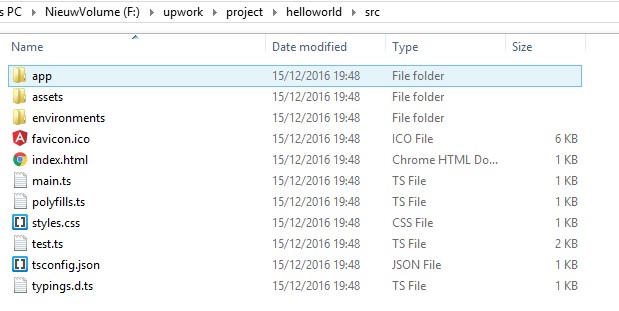
This command will create the main app component and relevant template along with typings, tsconfig, styles and e2e configuration.



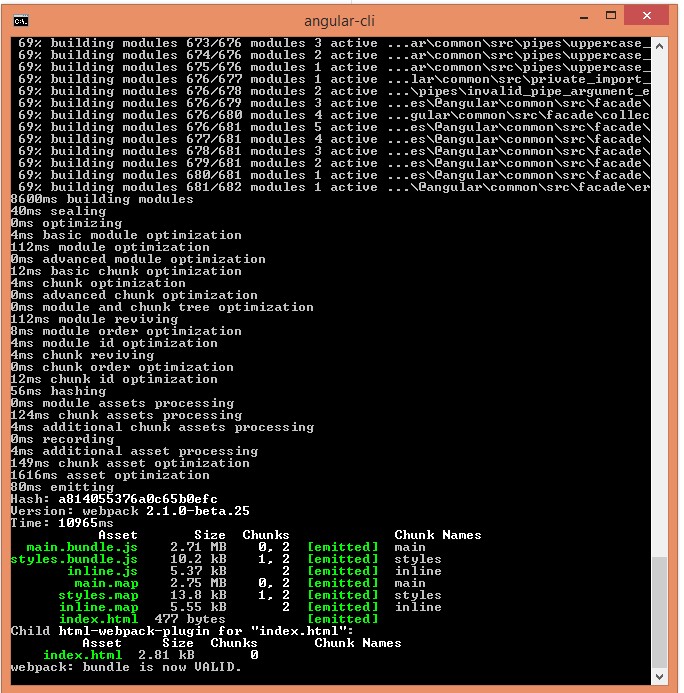
This step will also install all necessary required dependency library for the new project.Inside ‘helloworld’ project folder, there will be these following files created.



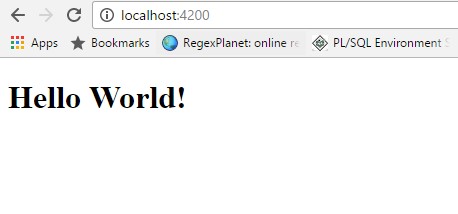
The source code for the application are inside ‘src’ folder as following:



1. The next step is to run the project with the command ‘ng serve’.
   1. Go to the ‘helloworld’ folder by running ‘cd helloworld’ command
   2. Run ‘ng serve’ in command prompt



1. From browser run ‘http://localhost:4200’and the output would be as following:



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

Rest API :--------------------------------------------

All Country :------

<https://restcountries.eu/rest/v2/all?fields=name;capital>

MongoDB: --------------------------------------------

Addfields:-------

<https://docs.mongodb.com/manual/reference/operator/aggregation/addFields/>

Angular 4 + Core2 +JWT :-----------------------------------------------

<https://angular-2-training-book.rangle.io/handout/features/typescript_classes.html>

<https://www.codeproject.com/Articles/1210559/Asp-net-core-Angular-Build-from-scratch-a-web-appl>

<https://fullstackmark.com/post/10/user-authentication-with-angular-and-asp-net-core>

UI Templates:--------------------------------------------------------------

<https://github.com/mrholek/CoreUI-Free-Bootstrap-Admin-Template>