**Javascript: (repl.it – Online Editor)**

[**https://stackblitz.com/**](https://stackblitz.com/)

[**https://jsfiddle.net/**](https://jsfiddle.net/)

[**https://playcode.io/**](https://playcode.io/)

[**https://codesandbox.io/**](https://codesandbox.io/)

**https://www.tutorialspoint.com/**

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

//Problem - Use Promises to resolve multiple functions with previouly returned result as input

//hi how are you

function greet(str) {

return str;

}

let p = new Promise((resolve, reject) => {

return resolve(greet("hi"));

})

.then((res) => {

return new Promise((resolve, reject) => {

return resolve(greet(res + " how"));

});

})

.then((res) => {

return new Promise((resolve, reject) => {

return resolve(greet(res + " are you"));

});

})

.then((res) => {

console.log(res);

});

**ES 2020**

* **globalThis**
* **Promise.allSettled([])**
* **?? it check only if property does not exist not value (p.name ?? ‘Mahesh’)**
* **?. Optional chaining operator**
* **BigInt**

**String:**

Slice(),substr(), indexOf(), search(), splice(), replace(), toUpperCase(),toLowerCase();

**Numbers:**

Integer to string 🡺 toString() ,

pasrseInt() string to integer

toFixed(1)

**Math**

Math.round(4.3) = 4,

Math.pow(10,2) =100

Math.sqrt(49) === Math.pow(49,0.5)

Math.abs(-123) =123, absolute value,

Math.ceil(3.4) = 4

Math.floor(3.4) = 3

Math.min(2,3,4,5)

Math.max(2,3,4,5)

**Array:**

Array.push()

Arry.pop()

Array.unshift()

Array.shift()

Arry.sort()

Arry.toString()

Arry.splice()

Delete arry[0]

Array1.concat(arry2)

Array.flat(); const matrix = [[1,1], [1,[[[[[[1]]]]]]]; => matrix.flat(Infinity); => flatArray is [1,1,1,1]

Array.flatMap();

Array.reduceRight();

Array.copyWithin();

Object.freeze(someObject);

It won’t allow to add new properties to such object, remove existing ones and change the current prototype of such object.

we still cannot change a property’s value if it’s an object:

Object.isFrozen(someObject)

## Object.seal()

With sealing, the situation is a little bit different. We still can’t add new properties, delete existing ones and every property becomes non-configurable but the values of properties can be changed:

Object.isSealed(sealedObject);

**Switch:**

**RegularExpression:** /asa/i

**DOM**

**BOM :** browser object model

[**https://www.thatjsdude.com/**](https://www.thatjsdude.com/)

console.time('myLoop');

for(var i=0; i<10000; i++){

2+2\*3;

}

console.timeEnd('myLoop');

document.body.contentEditable = true;

function myFunc() { var bar = function() { return 'world'; }; function foo() { return 'fooed'; } var a = 5; }

function myFunc() { function foo() { return 'fooed'; } var bar; var a; bar = function() { return 'world'; }; a = 5; }

**Javascript Algorithams**

1. **Fizz Buzz**



1. **HarmlessRansomNote**

**Big O Notation:** Big O Notation allows us to compare worst case performance of our algorithms in a standardized way

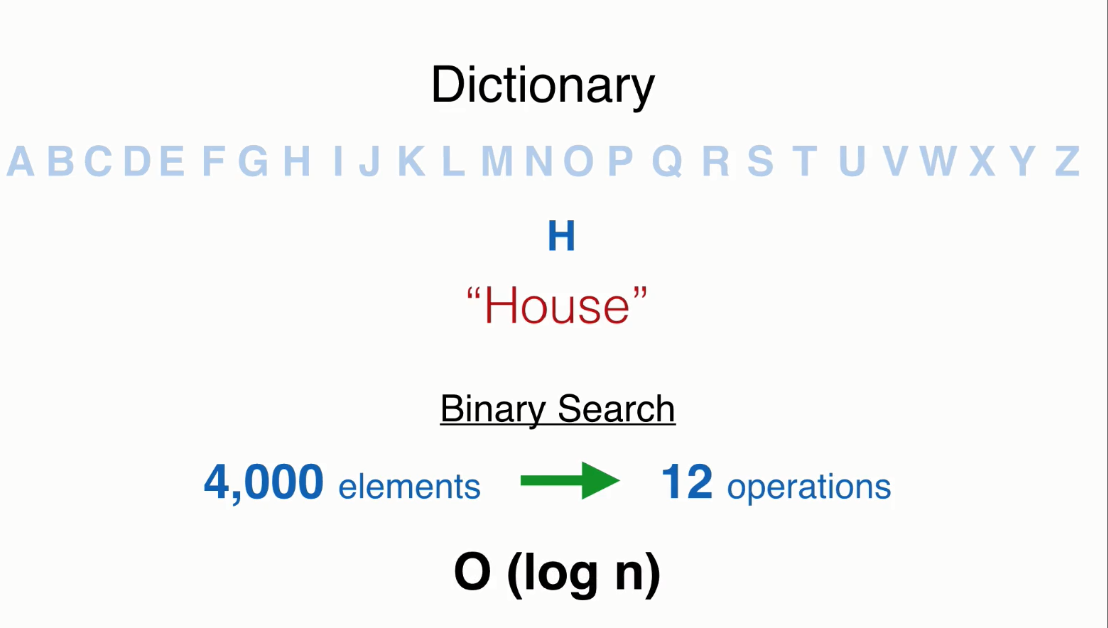
[**https://www.desmos.com/calculator**](https://www.desmos.com/calculator)

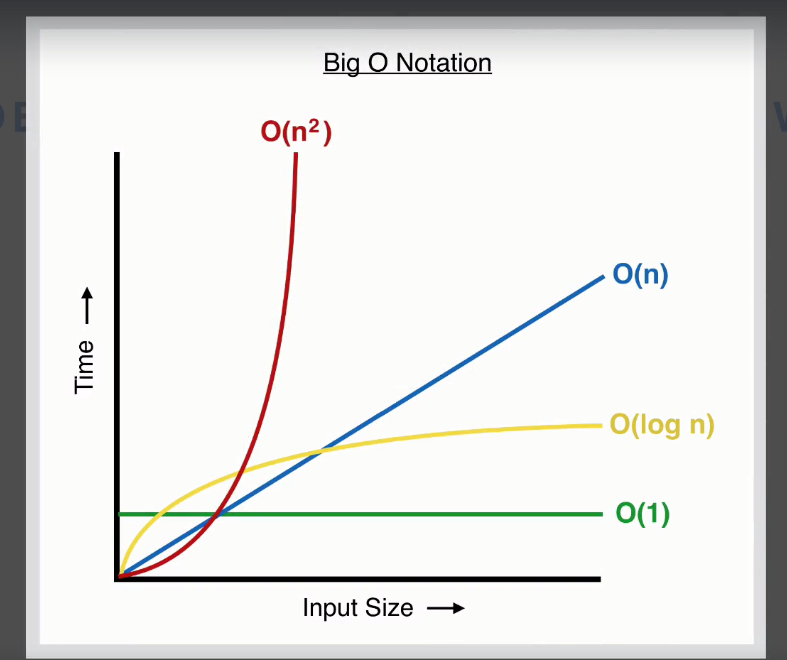
## O(1) – Constant Time Complexity

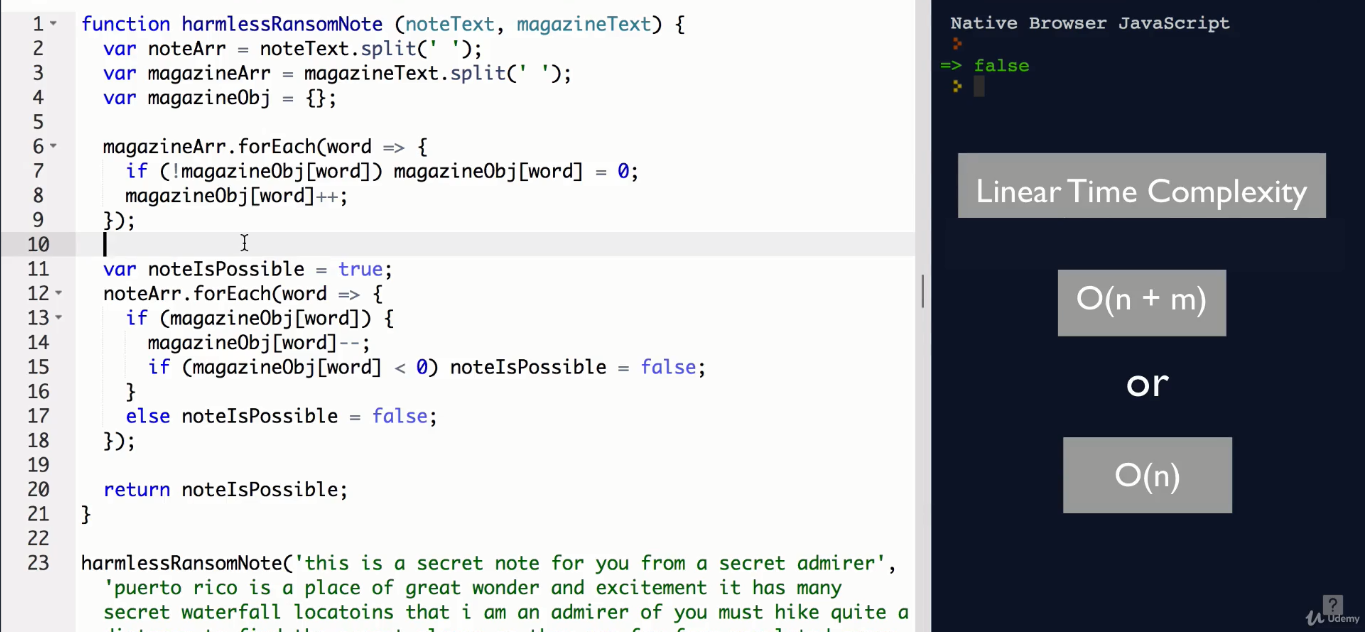
## O(log n) – Logarithmic

## O(n) – Linear

## O(n2) – Quadratic (Exponentional)



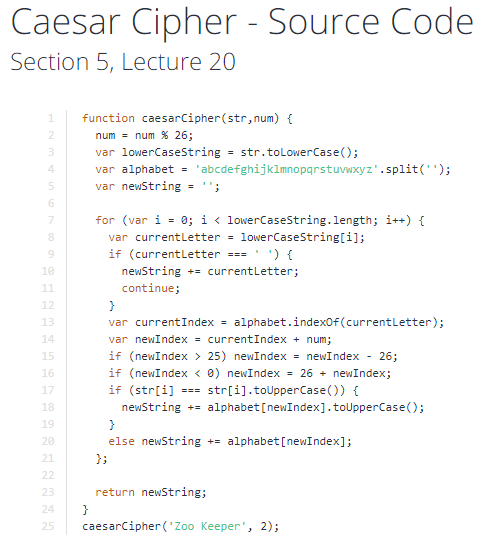




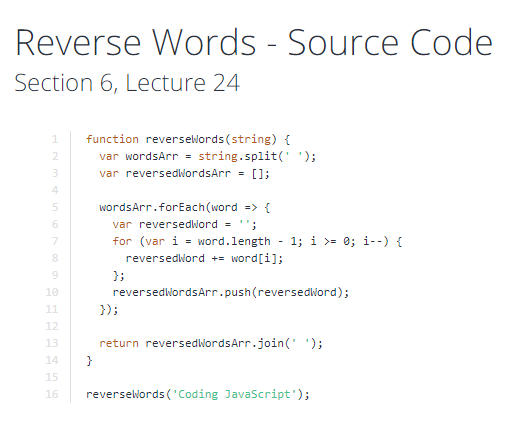
1. **Is Polindrome:**
2. function palindrome(str) {
3. var len = str.length;
4. for ( var i = 0; i < Math.floor(len/2); i++ ) {
5. if (str[i] !== str[len - 1 - i]) {
6. return false;
7. }
8. }
9. return true;
10. }

****

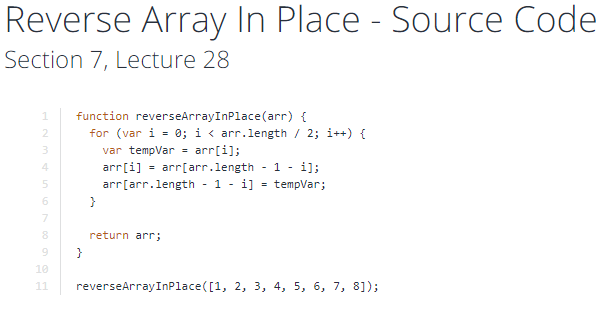
**4) caesarCipher:**

****

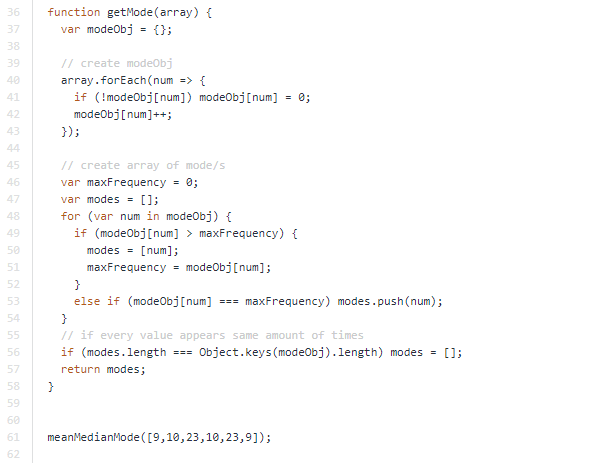
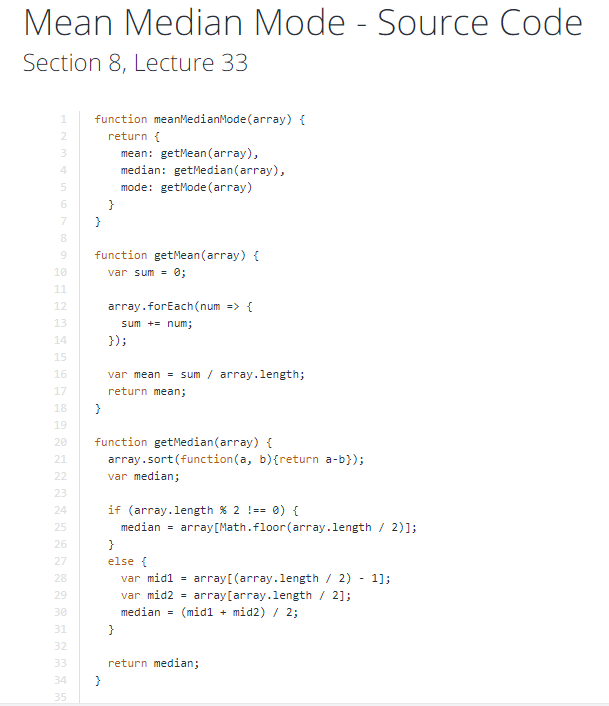
**5) Reverse Words without using reverse method**

****

**6) Reverse Array In Place:**

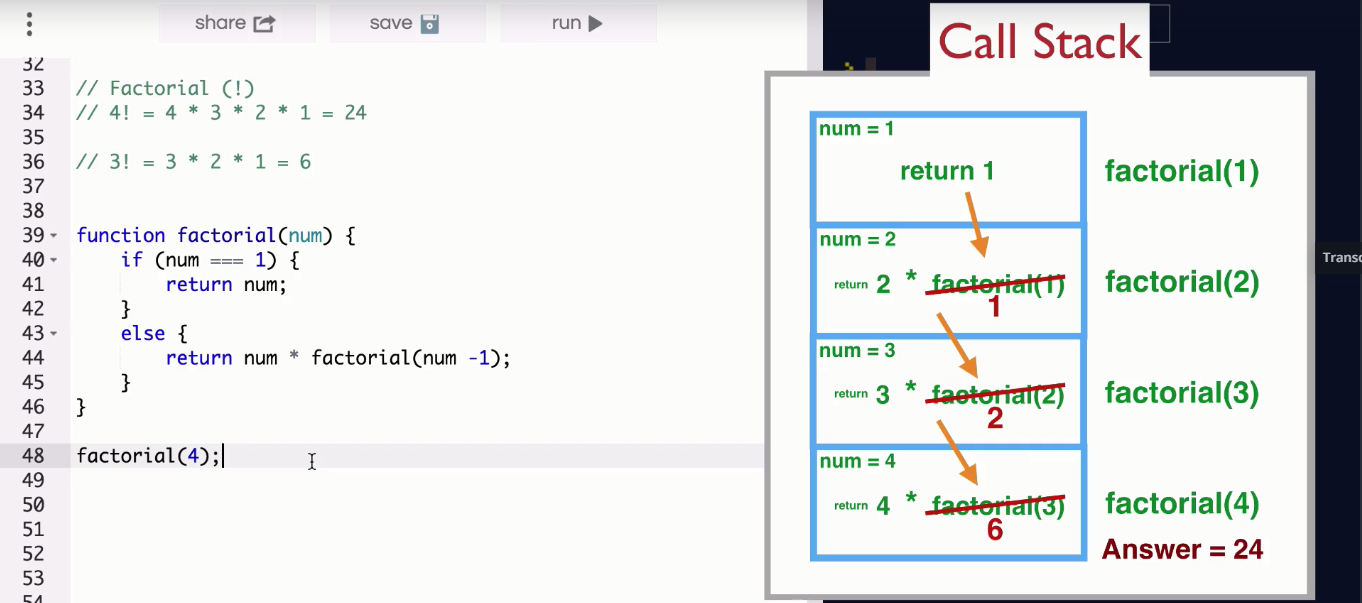
****

**7) Mean median Mode**

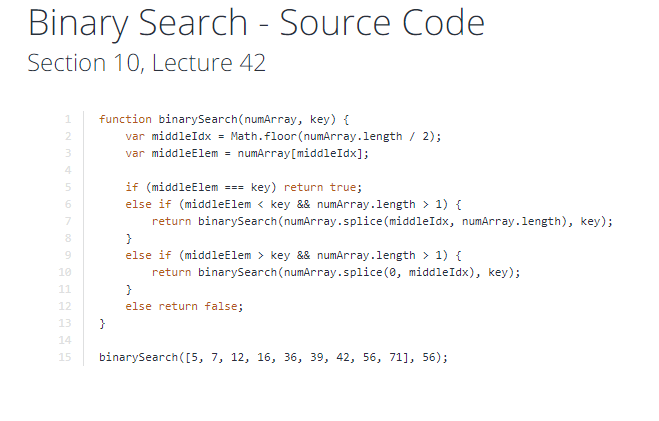
****

**8) Binary Search**

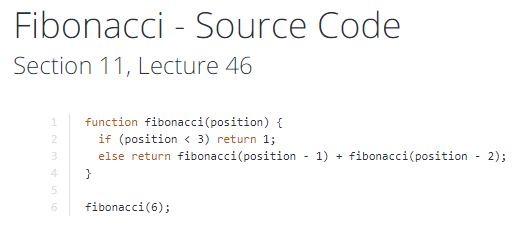
**1) Factorial –recursive function**

****

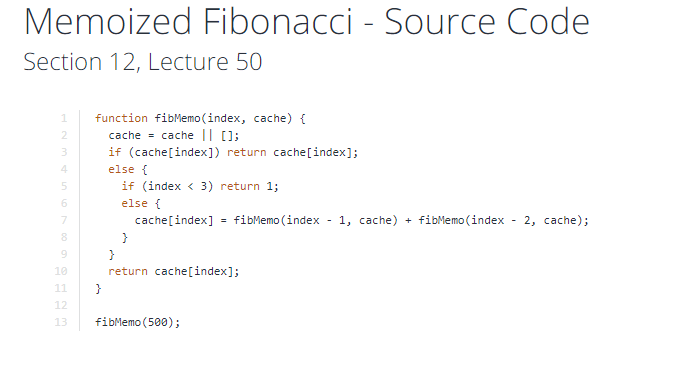
**Binary-search-example**

****

**9) Fibonacci**

****

**10) Memoized Fibonacci**

****

1) Output

# # # # #

# # # # \*

# # # \* \*

# # \* \* \*

# \* \* \* \*

\* \* \* \* \*

var i, j, k, str;

//outer loop

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

//inner loop

for(j=5; j>i; j--) {

str += ' #';

}

for(k=0; k<i; k++) {

str += ' \*';

}

console.log(str);

str = '';

}

## 2) Bitwise Operands

## 5 <<1 => 10 ; 0101 << 1 =>1010

## 3) Window Navigator

* navigator.appName
* navigator.appCodeName
* navigator.platform

# 4) Javascript Data Types

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| String | represents sequence of characters e.g. "hello" |
| Number | represents numeric values e.g. 100 |
| Boolean | represents boolean value either false or true |
| Undefined | represents undefined value |
| Null | represents null i.e. no value at all |

JavaScript non-primitive data types

The non-primitive data types are as follows:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| Object | represents instance through which we can access members |
| Array | represents group of similar values |
| RegExp | represents regular expression |

## 5) What is prototype in JavaScript? All the JavaScript objects has an object and its property called prototype & it is used to add and the custom functions and property. See Following example in which we create a property and function.

## 6) 1 + + “4” = 5; 1 - - “4” = 5; 1 – “4” = -3

## 7) shift,unshift,push,pop,slice,splice

## 8) Event babbles and captures

## 9) stack and quee

### 10) What is the difference between JavaScript and jscript?

a) Netscape provided the JavaScript language. Microsoft changed the name and called it JScript to avoid the trademark issue.In other words, you can say JScript is same as JavaScript, but it is provided by Microsoft.

### 11) What is the use of history object?

A) The history object of browser can be used to switch to history pages such as back and forward from current page or another page. There are three methods of history object.

1. history.back()
2. history.forward()
3. history.go(number): number may be positive for forward, negative for backward.

**12) There are 3 ways to create object in JavaScript.**

1. By object literal
2. By creating instance of Object
3. By Object Constructor

13) **typeof NAN and null, and differene between NULL and null**

**14)** **Client side JavaScript** comprises the basic language and predefined objects which are relevant to running java script in a browser. The client side JavaScript is embedded directly by in the HTML pages. This script is interpreted by the browser at run time.

**Server side JavaScript** also resembles like client side java script. It has relevant java script which is to run in a server. The server side JavaScript are deployed only after compilation.

### 15) In which location cookies are stored on the hard disk?

The storage of cookies on the hard disk depends on OS and the browser.

The Netscape navigator on Windows uses cookies.txt file that contains all the cookies. The path is : c:\Program Files\Netscape\Users\username\cookies.txt

The Internet Explorer stores the cookies on a file username@website.txt. The path is: c:\Windows\Cookies\username@Website.txt.

### 16 ) What is the difference between undefined value and null value?

**17)** **Bind, call and apply in javascript**

### 18) What is negative infinity? -2/0

### 19) What are the pop up boxes available in JavaScript?

A) Alert Box,Confirm Box,Prompt Box

20) canvas and svg images

21) this keyword in javascript

**22)JSON :** Javascript object notation, light weight object.

23) Jasnparse and JsonStringfy

25) cross browser in ajax call passing headers

26) eval in javascript 🡺 eval("1 + 2") //3; eval("var x = 2")// x: 2

27) Scope in js

28) Map in javascript

29) use strict, hoisting

30) return how it works

31) true + true: 2; true > false: true

32) Object.Keys(obj).length

33) hasOwnProperty is the only thing in JavaScript which deals with properties and does **not** traverse the prototype chain.

34) function myArrayMax(arr) {

var len = arr.length;

var max = -Infinity;

while (len--) {

if (arr[len] > max) {

max = arr[len];

}

}

return max;

}

# 35) JavaScript: Remove Duplicates from an Array

**const** names **=** ['John', 'Paul', 'George', 'Ringo', 'John'];

**let** unique **=** [...**new** Set(names)];

console.log(unique); *// 'John', 'Paul', 'George', 'Ringo'*

Another option is to use [filter()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter).

**const** names **=** ['John', 'Paul', 'George', 'Ringo', 'John'];

**let** x **=** (names) **=>** names.filter((item,index) **=>** names.indexOf(item) **===** index)

x(names); *// 'John', 'Paul', 'George', 'Ringo'*

## 36) Retrieve the duplicate values

**const** names **=** ['John', 'Paul', 'George', 'Ringo', 'John'];

**let** x **=** (names) **=>** names.filter((item,index) **=>** names.indexOf(item) **!==** index)

x(names); *// 'John'*

## Introduction

Observable and Promise both provide us with abstractions that help us deal with the asynchronous nature of applications.

### ****Promise****

Promises work with asynchronous operations. They either return a single value (i.e the promise resolves) or an error message (i.e the promise rejects).

Another important thing to remember regarding promises is that a request initiated from a promise is not canceled.

 Disadvantages of Promise:

1. User could not cancel a request to the API.
2. User could not retry a failed call.
3. As our application gets bigger, promises become hard to manage.

### ****Observable****

An Observable is an array or a sequence of events over time. It has at least two participants, the creator (the data source) and the subscriber (subscription where data is being consumed). Compared to a promise, an observable can be canceled.

RxJS is all about unifying the ideas of promise callbacks and data flow and making them easier to work with. Observables provide operators, like map, forEach, reduce...similar to an array.

There are also powerful operators like retry(),  reply(), retryWhen(), delay().

# Javascript event loop

**React**

**1) What’s the Challenges with React Native**

**A)** Working across separate iOS and Android codebases is challenging.

2) class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { /\* initial state \*/ };

}

}

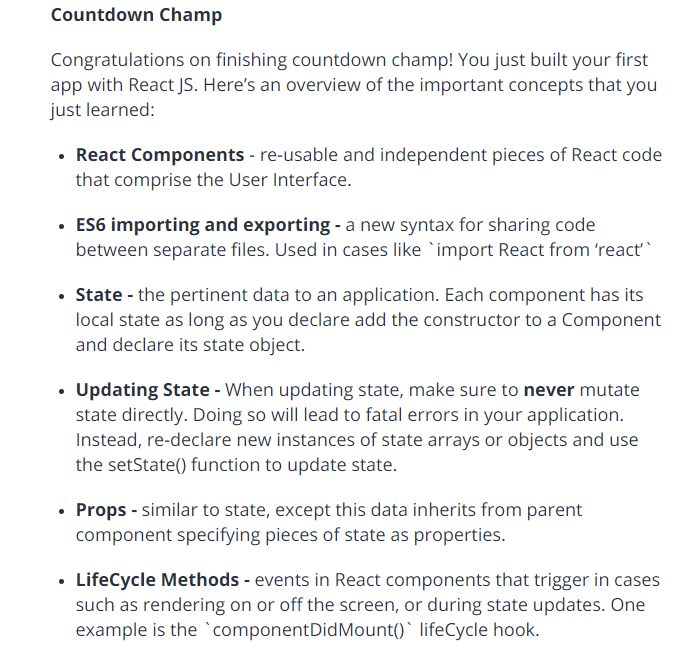
3) **what is a prop?**

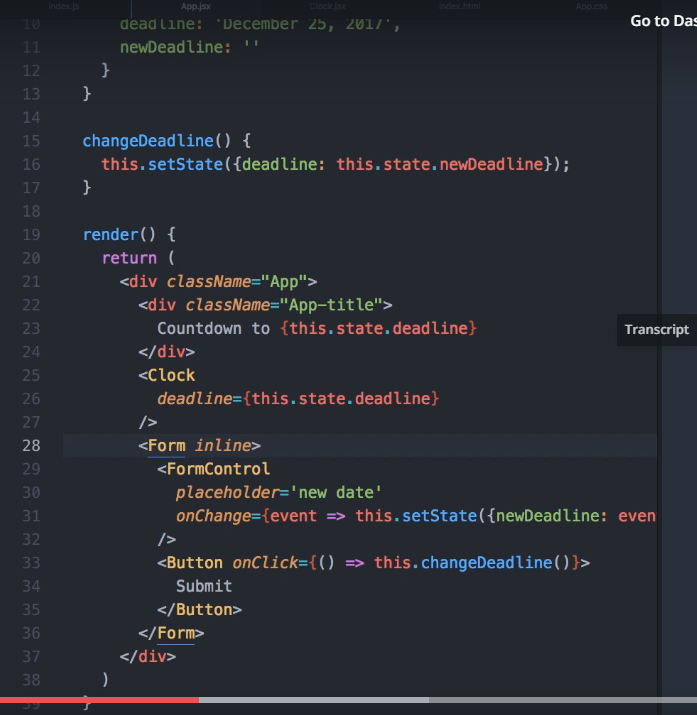
**4) pass data from one component to another component**

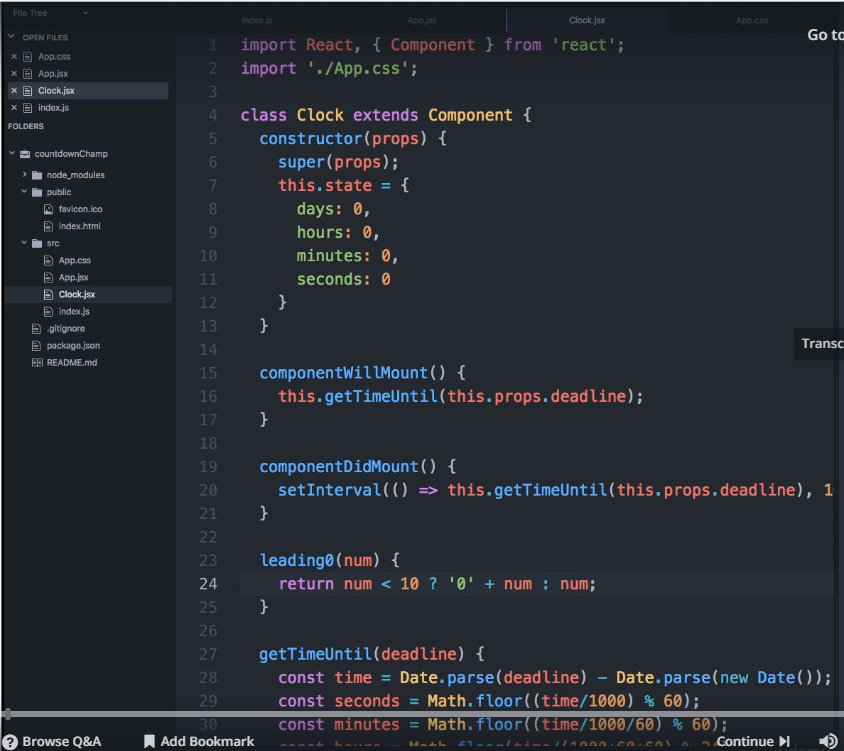
**5) How to create modal in react**

**Ract lifecycle**

* 1. **componentWillMount(){}**
  2. **componentDidMount(){}s**

****

****

****

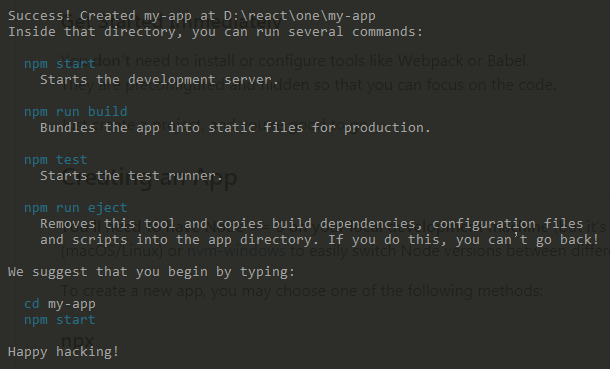
**Start the sample project**

<https://github.com/facebook/create-react-app>

npx create-react-app my-app

cd my-app

npm start



1. **React**
2. **React DOM**
3. **Always use super() inside the constructor of child component to access default properties of extend parent component**
4. **Render() {return (<div> all contnt </div>)}**
5. **Always use this.setState({key:value}) to update the state of component, never modify directly state of component**
6. **React re-call render() at the end of setState(), if you modify directly state then render method it won’t call directly.(wrong: “this.state.key =!this.state.key” correct: this.setState({key: !key}))**
7. **Don’t use methods as this.methid(), use like this.method**

**Interview main concepts in react and redux**

* **react version**
* **react life cycle**
* **react routing, and version**
* **react render map or foreach**
* **redux**
* **what basees redux work**
* **provider in react**
* **connect in react**
* **how can we skip the data rendering**
* **state manipulate, setState(), objectattaching**
* **spread operator**
* **combine multiple reducers**
* **check login any libraray**
* **redux devtools**
* **redux loger**
* **link and navlink difference**
* **arrrow**
* **babel and webpack**
* **code speltting**
* **jsx meaning**
* **contest in react**

## [Redux](http://www.mytectra.com/react-native-and-redux-course.html)

## 

## objectObdate.PNG

## redux.PNG

## **1.What are “actions” in Redux?**

## **2) What is the role of reducers in Redux?**

## **3) What is ‘Store’ in Redux?**

## **4) How is state changed in Redux?**

## **5)**

## **ES6**

## **Let**

## **const**

## **var**

## **arrow function**

## **delete**

## **css**

## **keyframes**

## **animation**

## **border radius**

## **backgroun iamges**

## **less/sacs**

## **first/last(psude)**

## **n,n+1**

## **position**

## **media queries**

## **boxmodal**

## ****Interview referred links****

## **Guru99**

## ****Angular2+:****

## **Setup**

## **Component**

## 

## ****Example:****

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

import { Iuser } from './iuser';

import { UsersService } from './users.service';

@Component({

selector: 'app-users',

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

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

})

export class UsersComponent implements OnInit {

users:Iuser[];

errorMessage:any;

constructor(private \_usersService:UsersService) { }

ngOnInit() {

this.getUsersList();

}

getUsersList(){

this.\_usersService

.getUserDetails()

.subscribe(

users => this.users = users,

error => this.errorMessage = error

);

}

}

## ****Module****

## ****Example:****

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

import { CommonModule } from '@angular/common';

import { BrowserModule } from '@angular/platform-browser';

import { HttpClientModule} from '@angular/common/http';

import { HttpModule } from '@angular/http';

import { UsersComponent } from './users.component';

import { UserRoutingModule } from './user-routing/user-routing.module';

import { UsersService } from './users.service';

import { LimitCharPipe } from '../shared/limit-char.pipe';

@NgModule({

imports: [

CommonModule,

BrowserModule,

HttpClientModule,

HttpModule,

UserRoutingModule

],

declarations: [ UsersComponent, LimitCharPipe ],

exports: [ UsersComponent ],

providers: [ UsersService ]

})

export class UsersModule { }

## ****Data Binding****

## **One way binding**

## **Two way binding**

## **Event binding**

## ****Directive****

## **NgIf**

## **NgFor**

## **NgClass**

## ****Pipe****

## **Uppercase**

## **Lowercase**

## **Decimal**

## **Currency**

## **Date**

## **Json**

## ****Example: custome pipe****

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

@Pipe({

name: 'limitChar' //we needs to use this in template

})

export class LimitCharPipe implements PipeTransform {

transform(value: any, args?: any): any {

if(value){

return (value.length > length ) ? value.substr(0, args) + '...': value;

}

}

}

## ****Service****

## ****Example:****

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

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

import { Observable } from 'rxjs/Observable';

import 'rxjs/add/operator/map';

import 'rxjs/add/operator/do';

import 'rxjs/add/operator/catch';

import 'rxjs/add/observable/throw';

import { Iuser } from './iuser';

@Injectable()

export class UsersService {

constructor(private \_http:Http) { }

getUserDetails(): Observable<Iuser[]>{

return this.\_http

.get('https://jsonplaceholder.typicode.com/posts')

.map((response:Response) => <Iuser[]> response.json())

.do(data => console.log('data = '+ data))

.catch(this.handlError);

}

private handlError(error:Response){

console.log(error);

let message =`Error status code ${error.status} at ${error.url}`;

return Observable.throw(message);

}

}

## **Class**

## **Guard**

## ****Interface****

## ****Example:****

export interface Iuser {

userId:number;

id:number;

body:string;

title:string;

}

## **Enum**

## **Module**

## **Input && Output**

## **Nested components**

## **@input / @output**

## **Custom events**

## ****Example:****

## ****Input:****

## inputs.PNG

## ****Output Example**:**

## output.PNG

## **Routing**

## ****Example** :**

## routing.PNG

## **Observals**

## **Subscriber**

## **Map**

## **Lifecycle hooks**

## ngOnChanges()- imp

## ngOnInit() - imp

## ngDoCheck()

## ngAfterContentInit()

## [ngAfterContentChecked()](https://angular.io/api/core/AfterContentChecked#ngAfterContentChecked)

## ngAfterViewInit()

## [ngAfterViewChecked()](https://angular.io/api/core/AfterViewChecked#ngAfterViewChecked)

## ngOnDestroy() – imp

## Canactivate method in routing time

## Query params

## ****Angular 4****

## **If else in template**

## **Animation splits out from core**

## **Ng-class**

## **Performance**

## **Routing upgraded**

## **Ng-template**

## **Email type added no need to write pattern to validate email**

## ****Canvas HTML5****

var canvas = document.getElementById('canvas');

var ctx = canvas.getContext('2d');

var w = canvas.width;

var h = canvas.height;

var ballsize = 20;

var ballcolor = '#ff0000';

var oldballcolor = '#000000';

var myball = { x: (ballsize + 5), y: (ballsize + 5) };

var movedir = { x: 5, y: 5 };

function draw() {

ctx.clearRect(0, 0, w, h);

ctx.beginPath();

ctx.arc(myball.x, myball.y, ballsize, 0, Math.PI \* 2, false);

var grid = ctx.createRadialGradient(200,100,20,200,100,300);

grid.addColorStop(0,ballcolor);

grid.addColorStop(1,oldballcolor);

ctx.fillStyle = grid;

ctx.shadowColor = '#000';

ctx.shadowBlur = 30;

ctx.shadowOffsetX = 20;

ctx.shadowOffesrY = 20;

ctx.fill();

}

function moveBall() {

var ramNumber = Math.floor((Math.random()\*30) + 10);

if(myball.y >= (h-ballsize) || myball.y <= ballsize) {

movedir.y = -movedir.y;

// ballsize = ramNumber;

oldballcolor = ballcolor;

ballcolor = '#'+Math.random().toString(16).substr(-6);

}

if(myball.x >= (w-ballsize) || myball.x <= ballsize) {

movedir.x \*= -1;

// ballsize = ramNumber;

oldballcolor = ballcolor;

ballcolor = '#'+Math.random().toString(16).substr(-6);

}

myball.x += movedir.x;

myball.y += movedir.y;

draw();

requestAnimationFrame(moveBall);

}

moveBall();

## Get canvas data in

var dataURL = canvas.toDataURL();

## Node JS:

## Node Js and Chrome both uses the same v8 engines inside to run the scripts

## Node is not a programming language.

## Node is not a Framework.

## Node is a runtime environment for executing Java script code

## Node is highly scalable, data-intensive and real-time apps because it’s Non - Blocking of It’s asynchronous nature.

## It’s single thread to handle multiple requests

## Node applications are asynchronous by default

## Do not use Node for CPU-intensive applications only use for data-intensive application

## Window object is not available because it’s not running on browser

## Node modules:

## Operating System

## File System

## Events

## Http

## In node global is equal to window

## Every file in node considered as module

## Module.exports = {}; we will add all methods and variable to exports from one module to another module

## We will use require to load another file/module

## Events:

## 

## Angular 6

## Form Types

## Template driven

## Reactive forms

## Form Groups()

## FormControles()

## Neseted FormGroups()

## setValues(), PatchValues() – To update values

## Needs to know about touch, dirty, prism, valid and invalid

## FormBuilder

## Groups()

## Controls()

## Array[]

## FormValidators

## Required, min, max,minlength,maxlength,requiredTo,email, pattern == Builtin Validators

## We can add custome validators

## valueChanges.subscribe();

## setValidators()

## clearValidators()

## updateValueAndValidity()

## OOPS:

## ****Inheritance or**** Code Reuse patterns (objects can inherit features from other objects)

## ****Polymorphism**** (objects can share the same interface—how they are accessed and used—while their underlying implementation of the interface may differ)

## ****Encapsulation**** or Object Creation patterns (each object is responsible for specific tasks).

## we are concerned with only Inheritance and Encapsulation since only these two concepts apply to OOP in JavaScript

## JavaScript-Garden

## [http://www.thatjsdude.com](http://www.thatjsdude.com/)

## <https://www.geeksforgeeks.org/>

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

## Differences between Angular 2/4/5/6/7/8/9

## What’s New In Angular 9:

1. Angular 9 Released October/November 2019
2. Added undecorated classes migration schematic in the core.
3. The formControlName also accepts a number in the form.
4. Update rxjs peerDependencies minimum requirement
5. Support ng-add in localize package.
6. Allow retrieving synchronized analyzed NgModules
7. Enable logging on TypeScriptHost for language-service

**What’s New In Angular 8:**

* + - 1. Angular 8 Released April/May 2019
      2. Ivy: Next-gen Renderer for Angular Framework
         1. Shipment of pre-compiled code
         2. Compilation without the need of Metadata.json
         3. Meta programming
      3. Added Web Workers
      4. Added Lazy Loading
      5. Improvement in ng-upgrade
      6. Added Support for Node 10
      7. Improvements on CLI workflow
      8. Upgrading Angular Material
      9. Added TypeScript 3.4 support
      10. Added Differential Modern JavaScript Loading
      11. Improved Web Worker Bundling
      12. Added a Navigation Type Available during Navigation in the Router
      13. Added pathParamsOrQueryParamsChange mode for runGuardsAndResolvers in the Router
      14. Allow passing state to routerLink Directives in the Router
      15. Allow passing state to NavigationExtras in the Router
      16. Restore whole object when navigating back to a page managed by Angular Router
      17. Added support for SASS
      18. Resolve generated Sass/Less files to .css inputs
      19. Added Angular Router Backward Compatibility
      20. Enhanced Web Worker Bundling

**What’s New In Angular 7:**

1. Angular 7 Released on October 2018
2. This is a major release and expanding to the entire platform including-
   1. Core framework,
   2. Angular Material,
   3. CLI
3. Added a new interface - UrlSegment[] to CanLoad interface
4. Added a new interface -  DoBootstrap interface
5. Angular 7 added a new compiler - Compatibility Compiler (ngcc)
6. Introduce a new Pipe called - KeyValuePipe
7. Angular 7 now supporting to TypeScript 3.1
8. Added Virtual Scrolling features
9. Added Drag & Drop features
10. Added Better Error Handling
11. Added a new elements features - enable Shadow DOM v1 and slots
12. Added a new router features - warn if navigation triggered outside Angular zone
13. Added a new mapping for ngfactory and ngsummary files to their module names in AOT summary resolver.
14. Added a new "original" placeholder value on extracted XMB
15. Added a new ability to recover from malformed URLs
16. Added a new compiler support dot (.) in import statements and also avoid a crash in ngc-wrapped
17. Update compiler to flatten nested template fns
18. Added Native Script
19. Added Bundle Budget

**What’s New In Angular 6:**

1. Angular 6 Released on April 2018
2. Added Angular Element
3. Added Service worker
4. Added Internationalization (i18n)
5. Added Bazel Compiler
6. Added New commons - ng-add / ng-update
7. Added Model change events - ngModelChange
8. Added TypeScript 2.7 support
9. Improved decorator error messages
10. <ng-template> updated to <template>

**What’s New Angular 5?**

1. Include Representation of Placeholders to xliff and xmb in the compiler
2. Include an Options Arg to Abstract Controls in the forms controls
3. Include add default updateOn values for groups and arrays to form controls
4. Include updateOn blur option to form controls
5. Include updateOn submit option to form controls
6. Include an Events Tracking Activation of Individual Routes
7. Include NgTemplateOutlet API as stable in the common controls
8. Create StaticInjector which does not depend on Reflect polyfill
9. Include [@.disabled] attribute to disable animation children in the animations
10. Make AOT the default
11. Watch mode
12. Type checking in templates
13. More flexible metadata
14. Remove \*.ngfactory.ts files
15. Better error messages
16. Smooth upgrades
17. Tree-Shakeable components
18. Hybrid Upgrade Application

**What’s New Angular 4?**

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

## Angular Questions

## Q1: Explain the difference between Promise and Observable in Angular?

**Answer:**  
**Promises**:

* return a single value
* not cancellable
* more readable code with try/catch and async/await

**Observables**:

* work with multiple values over time
* cancellable
* support map, filter, reduce and similar operators
* use Reactive Extensions (RxJS)
* an array whose items arrive asynchronously over time

**Q2: Why should ngOnInit be used, if we already have a constructor?**

**Answer:**

* The Constructor is a default method of the class that is executed when the class is instantiated and ensures proper initialization of fields in the class and its subclasses.
* ngOnInit is a life cycle hook called by Angular2 to indicate that Angular is done creating the component.
* Mostly we use ngOnInit for all the initialization/declaration and avoid stuff to work in the constructor. The constructor should only be used to initialize class members but shouldn't do actual "work". So you should use constructor() to setup Dependency Injection and not much else. ngOnInit() is better place to "start" - it's where/when components' bindings are resolved.

**Q3: What is AOT?**

**Answer:**  
The Angular Ahead-of-Time compiler pre-compiles application components and their templates during the build process. Apps compiled with AOT launch faster for several reasons.

* Application components execute immediately, without client-side compilation.
* Templates are embedded as code within their components so there is no client-side request for template files.
* You don't download the Angular compiler, which is pretty big on its own.
* The compiler discards unused Angular directives that a tree-shaking tool can then exclude.

## Q5: What is the purpose of Wildcard route?

**Answer:**  
If the URL doesn't match any predefined routes then it causes the router to throw an error and crash the app. In this case, you can use wildcard route. A wildcard route has a path consisting of two asterisks to match every URL.

For example, you can define PageNotFoundComponent for wildcard route as below

{ path: '\*\*', component: PageNotFoundComponent }

## Q6: What are the utility functions provided by RxJS?

**Answer:**  
The RxJS library also provides below utility functions for creating and working with observables.

1. Converting existing code for async operations into observables
2. Iterating through the values in a stream
3. Mapping values to different types
4. Filtering streams
5. Composing multiple streams

## Q7: What is subscribing?

**Answer:**  
An Observable instance begins publishing values only when someone subscribes to it. So you need to subscribe by calling the **subscribe()** method of the instance, passing an observer object to receive the notifications.

Let's take an example of creating and subscribing to a simple observable, with an observer that logs the received message to the console.

Creates an observable sequence of 5 integers, starting from 1

const source = range(1, 5);

// Create observer object

const myObserver = {

next: x => console.log('Observer got a next value: ' + x),

error: err => console.error('Observer got an error: ' + err),

complete: () => console.log('Observer got a complete notification'),

};

// Execute with the observer object and Prints out each item

myObservable.subscribe(myObserver);

// => Observer got a next value: 1

// => Observer got a next value: 2

// => Observer got a next value: 3

// => Observer got a next value: 4

// => Observer got a next value: 5

// => Observer got a complete notification

## Q9: What's new in Angular 8?

**Answer:**  
This release is mostly about Ivy and the possibility to give it a try, but it also includes a few features and breaking changes, namely:

* **Differential loading** - with differential loading, two bundles are created when building for production: a bundle for modern browsers that support ES2015+ and a bundle for older browsers that only support the ES5 version of JavaScript
* **TypeScript 3.4** support
* **Ivy** - it is the new compiler/runtime of Angular. It will enable very cool features in the future, but it is currently focused on not breaking existing applications.
* **Bazel** support - it is a build tool developed and massively used by Google, as it can build pretty much any language.
* **Lazy-loading with import()** syntax
* // from
* loadChildren: './admin/admin.module#AdminModule'
* // to

loadChildren: () => import('./races/races.module').then(m => m.RacesModule)

* To help people migrating from AngularJS, a bunch of things have been added to the **location services** in Angular
* The **service worker registration** has a new option that allows to specify when the registration should take place.
* @angular/http has been removed from 8.0, after being replaced by @angular/common/http in 4.3 and officially deprecated in 5.0,

## Q9: Angular 8: What is Bazel?

**Answer:**  
Google open sourced the software responsible for building most of its projects under the name **Bazel**. Bazel is a powerful tool which can keep track of the dependencies between different packages and build targets.

Some of the features of Bazel are:

* It has a smart algorithm for determining the build dependencies - based on the dependency graph of a project, Bazel determines which targets it can build in parallel
* Bazel is independent of the tech stack. We can build anything we want with it using the same interface. For example, there are plugins for Java, Go, TypeScript, JavaScript, and more

## Q10:Angular 8: What is Angular Ivy?

**Answer:**  
A big part of Angular is its compiler: it takes all your HTML and generates the necessary JS code. This compiler (and the runtime) has been completely rewritten over the last year, and this is what Ivy is about. The last rewrite was done in Angular 4.0.

**Ivy** is a complete rewrite of the compiler (and runtime) in order to:

* reach better build times (with a more incremental compilation)
* reach better build sizes (with a generated code more compatible with tree-shaking)
* unlock new potential features (metaprogramming or higher order components, lazy loading of component instead of modules, a new change detection system not based on zone.js…)

## Q12: Angular 8: Explain Lazy Loading in Angular 8?

**Answer:**  
Lazy loading is one of the most useful concepts of Angular Routing and brings down the size of large files. This is done by lazily loading the files that are required *occasionally*.

Angular 8 comes up with support for **dynamic imports** in our router configuration. This means that we use the import statement for lazy loading the module and this will be understood by the IDEs, webpack, etc.

Angular 7:

{path: ‘user’, loadChildren: ‘./users/user.module#UserModule’}

Angular 8:

{path: ‘user’, loadChildren: () => import(‘./users/user.module’).then(m => m.UserModule)};

New with Angular 8, loadChildren expects a function that uses the dynamic import syntax to import your lazy-loaded module only when it’s needed. As you can see, the dynamic import is promise-based and gives you access to the module, where the module’s class can be called

## Q13: How to detect a route change in Angular?

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

import { Router, Event, NavigationStart, NavigationEnd, NavigationError } from '@angular/router';

@Component({

selector: 'app-root',

template: `<router-outlet></router-outlet>`

})

export class AppComponent {

constructor(private router: Router) {

this.router.events.subscribe((event: Event) => {

if (event instanceof NavigationStart) {

// Show loading indicator

}

if (event instanceof NavigationEnd) {

// Hide loading indicator

}

if (event instanceof NavigationError) {

// Hide loading indicator

// Present error to user

console.log(event.error);

}

});

}

}

## What is Zone in Angular?

## Updates the view with modified data

## Explain the purpose of Service Workers in Angular ?

## It works with cache try to load files from cache.

## **Question: Please explain the various features of Angular.**

**Answer:** There are several features of Angular that makes it an ideal front end JavaScript framework. Most important of them are described as follows:

* **Accessibility Applications**

Angular allows creating accessible applications using ARIA-enabled components, built-in a11y test infrastructure, and developer guides.

* **Angular CLI**

Angular provides support for command-line interface tools. These tools can be used for adding components, testing, instant deploying, etc.

* **Animation Support**

Angular’s intuitive API allows the creation of high-performance, complex animation timelines with very little code.

* **Cross-Platform App Development**

Angular can be used for building an efficient and powerful desktop, native, and progressive web apps. Angular provides support for building native mobile applications using Cordova, Ionic, or NativeScript.

Angular allows creating high performance, offline, and zero-step installation progressive web apps using modern web platform capabilities. The popular JS framework can also be used for building desktop apps for Linux, macOS, and Windows.

* **Code Generation**

Angular is able to convert templates into highly-optimized code for modern JavaScript virtual machines.

* **Code Splitting**

With the new Component Router, Angular apps load quickly. The Component Router offers automatic code-splitting so that only the code required to render the view that is requested by a user is loaded.

* **Synergy with Popular Code Editors and IDEs**

Angular offers code completion, instant errors, etc. with popular source code editors and IDEs.

* **Templates**

Allows creating UI views with a simple and powerful template syntax.

* **Testing**

Angular lets you carry out frequent unit tests using Karma. The Protractor allows running faster scenario tests in a stable way.

**Question: Discuss the advantages and disadvantages of using Angular?**  
**Answer:** Following are the various advantages of using Angular:

* Ability to add a custom directive
* Exceptional community support
* Facilitates client and server communication
* Features strong features, such as Animation and Event Handlers
* Follows the MVC pattern architecture
* Offers support for static template and Angular template
* Support for two-way data-binding
* Supports dependency injection, RESTful services, and validations

Disadvantages of using Angular are enumerated as follows:

* Complex SPAs can be inconvenient and laggy to use due to their size
* Dynamic applications do not always perform well
* Learning Angular requires a decent effort and time

**Question: What is string interpolation in Angular?**

**Answer:** Also referred to as moustache syntax, string interpolation in Angular refers to a special type of syntax that makes use of template expressions in order to display the component data. These template expressions are enclosed within double curly braces i.e. {{ }}.

**Question**: **Explain Angular Authentication and Authorization**

**Get Token - Authentication**

## Pass token to all API’s **Authorization**

**Question**: **What are the building blocks of Angular?**  
**Answer**: There are essentially 9 building blocks of an Angular application. These are:

1. **Components –** A component controls one or more views. Each view is some specific section of the screen. Every Angular application has at least one component, known as the [root component](https://www.learnhowtoprogram.com/javascript/angular/angular-2-setup-root-component-root-module-and-more). It is bootstrapped inside the main module, known as the root module. A component contains application logic defined inside a class. This class is responsible for interacting with the view via an API of properties and methods.
2. **Data Binding –** The mechanism by which parts of a template coordinates with parts of a component is known as data binding. In order to let Angular know how to connect both sides (template and its component), the binding markup is added to the template HTML.
3. **Dependency Injection (DI) –** Angular makes use of DI to provide required dependencies to new components. Typically, dependencies required by a component are services. A component’s constructor parameters tell Angular about the services that a component requires. So, a dependency injection offers a way to supply fully-formed dependencies required by a new instance of a class.
4. **Directives –** The templates used by Angular are dynamic in nature. Directives are responsible for instructing Angular about how to transform the DOM when rendering a template. Actually, components are directives with a template. Other [types of directives](https://angular.io/guide/attribute-directives) are attribute and structural directives.
5. **Metadata –** In order to let Angular know how to process a class, metadata is attached to the class. For doing so decorators are used.
6. **Modules –** Also known as NgModules, a module is an organized block of code with a specific set of capabilities. It has a specific application domain or a workflow. Like components, any Angular application has at least one module. This is known as the root module. Typically, an Angular application has several modules.
7. **Routing –** An Angular router is responsible for interpreting a browser URL as an instruction to navigate to a client-generated view. The router is bound to links on a page to tell Angular to navigate the application view when a user clicks on it.
8. **Services –** A very broad category, a service can be anything ranging from a value and function to a feature that is required by an Angular app. Technically, a service is a class with a well-defined purpose.
9. **Template –** Each component’s view is associated with its companion template. A template in Angular is a form of HTML tags that lets Angular know that how it is meant to render the component.

## 

## **Question**: **What is AOT (Ahead-Of-Time) Compilation?** ****Answer****: Each Angular app gets compiled internally. The Angular compiler takes in the JS code, compiles it and then produces some JS code. This happens only once per occasion per user. It is known as AOT (Ahead-Of-Time) compilation.

**Question: Could you explain the various types of filters in AngularJS.**  
**Answer:** In order to format the value of expression so that it can be displayed to the user, AngularJS has filters. It is possible to add these filters to the controllers, directives, services, or templates. AngularJS also provides support for creating custom filters.

Organizing data in such a way so that it is displayed only when certain criteria are fulfilled is made possible using filters. Filters are added to the expressions using the pipe ‘|’ character. Various types of AngularJS filters are enumerated as follows:

* *currency* – Formats a number to the currency format
* *date* – Formats a data to some specific format
* *filter* – Selects a subset of items from an array
* *json* – Formats an object to a JSON string
* *limitTo* – Limits an array or string into a specified number of characters or elements
* *lowercase* – Formats a string to lowercase
* *number* – Formats a number to a string
* *orderBy* – Orders an array by an expression

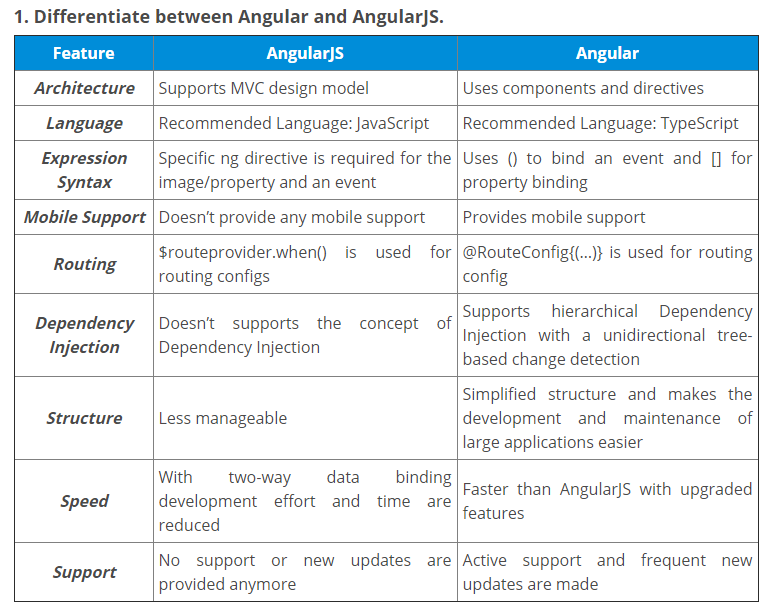
**Question: Discuss the lifecycle designed for directive and components in Angular JS especially for the newly introduced version 6.0?**Answer:

Components and directive of Angular JS follow the following typical lifecycle.

* ngOnInit
* ngDoCheck
* ngOnDestroy
* Constructor
* ngOnChanges
* ngAfterContentInit (only for components)
* ngAfterContentChecked (only for components)
* ngAfterViewInit (only for components)
* ngAfterViewChecked (only for components)

## ****What is the difference between an Annotation and a Decorator in Angular?****

Annotations in angular are “only” metadata set of the class using the Reflect Metadata library. They are used to create an “annotation” array. On the other hand, decorators are the design patterns that are used for separating decoration or modification of a class without actually altering the original source code



hostlistener-and-hostbinding/



[Q. 16.What is runGuardsAndResolvers in Angular 8?](https://www.bestinterviewquestion.com/question/what-is-runguardsandresolvers-in-angular-8-ryzvo4523w0)

**Angular 8** introduced a number of new and unique options to **runGuardsAndResolvers**. In general, **runGuardsAndResolvers**is an option which is used for the Angular router configuration in order to control the resolvers and guards. The first option available in **runGuardsAndResolvers**is pathParamsChange. Through, this option router will re-run the guards and resolvers. Whenever you want to control over the resolvers and guards, use **runGuardsAndResolvers**option in Angular 8.

**Angular Unit Tests**

Tests are the best way to prevent software defects.

**Karma**

Karma is a direct product of the AngularJS team from struggling to test their own framework features with existing tools. As a result of this, they made Karma and have transitioned it to Angular as the default test runner for applications created with the Angular CLI.

**Jasmine**

Jasmine is a behavior-driven development framework for testing JavaScript code that plays very well with Karma. Jasmine is also dependency free and doesn’t require a DOM

// A Jasmine suite

describe('Adder', () => {

// A jasmine spec

it('should be able to add two whole numbers', () => {

expect(Adder.add(2, 2)).toEqual(4);

});

it('should be able to add a whole number and a negative number', () => {

expect(Adder.add(2, -1)).toEqual(1);

});

it('should be able to add a whole number and a zero', () => {

expect(Adder.add(2, 0)).toEqual(2);

});

});

import { async, TestBed } from '@angular/core/testing';

import { RouterTestingModule } from '@angular/router/testing';

import { AppComponent } from './app.component';

describe('AppComponent', () => {

beforeEach(async(() => {

TestBed.configureTestingModule({

imports: [

RouterTestingModule

],

declarations: [

AppComponent

],

}).compileComponents();

}));

it('should create the app', async(() => {

const fixture = TestBed.createComponent(AppComponent);

const app = fixture.debugElement.componentInstance;

expect(app).toBeTruthy();

}));

});

## [Testing an Angular service](https://scotch.io/tutorials/testing-angular-with-jasmine-and-karma-part-1#toc-testing-an-angular-service)

I personally find injecting dependencies within it() to be a bit repetitive and harder to read as it's done in the default scaffolding for our test file as shown below:

it('should be created', inject([TestService], (service: TestService) => {

expect(service).toBeTruthy();

}));

With a few minor changes, we can move this into the beforeEach removing the duplication from each it.

import { TestBed } from '@angular/core/testing';

import { UsersService } from './users.service';

describe('UsersService', () => {

let usersService: UsersService; // Add this

beforeEach(() => {

TestBed.configureTestingModule({

providers: [UsersService]

});

usersService = TestBed.get(UsersService); // Add this

});

it('should be created', () => { // Remove inject()

expect(usersService).toBeTruthy();

});

});

In the code above, TestBed.configureTestingModule({}) sets up the service we want to test with UsersService set in providers. We then inject the service into our test suite using TestBed.get() with the service we want to test as the argument. We set the return value to our local usersService variable which will allow us to interact with this service within our tests just as we would within a component.

Now that our test setup is restructured, we can add a test for an all method which will return a collection of users.





import { of } from 'rxjs'; // Add import

describe('UsersService', () => {

...

it('should be created', () => {

expect(usersService).toBeTruthy();

});

// Add tests for all() method

describe('all', () => {

it('should return a collection of users', () => {

const userResponse = [

{

id: '1',

name: 'Jane',

role: 'Designer',

pokemon: 'Blastoise'

},

{

id: '2',

name: 'Bob',

role: 'Developer',

pokemon: 'Charizard'

}

];

let response;

spyOn(usersService, 'all').and.returnValue(of(userResponse));

usersService.all().subscribe(res => {

response = res;

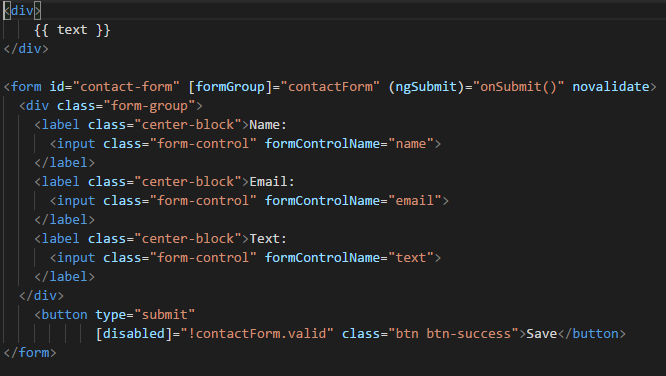
});

expect(response).toEqual(userResponse);

});

});

});







**Bootstrap**

**Q ) Define Bootstrap.**

**Answer:** Bootstrap is a front-end framework that is used for creating HTML, CSS, and JS web applications. Its layout is very responsive, easy and fast to use. It mostly focuses on building a mobile application with having design templates for creating UI like Dropdown, Forms, Buttons, Alerts Tab, etc.

**Q**: **What do you mean by the Bootstrap Grid System?**  
**A**: The Bootstrap Grid System is a responsive, mobile-first system that scales up to 12 columns as per the increase in the device or viewport size. The system features predefined classes for easy layout options and powerful mix-ins for generating effectively semantic layouts.

**Q ) Explain the features of Bootstrap.**

**Answer: Its Features include:**

* Open Source for use
* Compatibility with all browsers.
* Responsive designs
* Easy to use and fast.

**Q ) How many types of layouts are there in Bootstrap?**

**Answer:** There are two types of layouts in Bootstrap.

**They are:**

* **Fluid Layout** : Fluid Layout is useful when you need to make an app which involves the full width of the screen i.e. Fluid Layout adjusts itself according to the browser size.
* **Fixed Layout:** Fixed layout is responsive and easy to use but just like the fluid layout, it cannot adjust itself according to the browser size. Fixed Layout should be 940 px in most cases.

**Q ) When will you use <code>tag and <pre>tag?**

**Answer:** <code>tag is used to show the code inline and <pre>tag is used to show code with multiple lines

**Q ) What is a progress bar in bootstrap?**

**Answer:** Progress bar is used with HTML tag style in HTML element using <**progress**> keyword. In bootstrap we used html5 <**progress**> with CSS classes that have special features in bootstrap, that is only made for the progress bar.

**Q**#**13) Name the contextual classes that are used with the progressive bar in bootstrap.**

**Answer:** The contextual classes used with progressive bar are as follows.

* Progress-success
* Progress-info
* Progress-warning
* Progress-danger

**Q** #**15) What are the different button styles in Bootstrap?**

**Answer:** In bootstrap there are seven styles which we can use with the bootstrap button.

* .btn-default.
* .btn-primary
* .btn-success
* .btn-info
* .btn-warning.
* .btn-danger.
* .btn-link.

**Q**#**16) What are Bootstrap alerts?**

**Answer:** This is used to create presume alert messages, which adds style to the messages to look more noticeable to the user.

There are four classes in alerts i.e .alert-success, .alert-info, .alert-warning, .alert-danger.

**Q**#**17) Define Bootstrap thumbnails.**

**Answer:** It is a way to use the layout images, videos, text etc. in a grid system. We can create thumbnails by adding a tag with the class **.thumbnails**around the image.

This will add four pixels of padding and a grey border.

**Q**#**18) Explain Modal plugin in Bootstrap.**

**Answer:** A model is an inherited window that is layered over its parent window. This is used to augment the user experience and adds different functionalities to the users.

Model windows are created with the help of the modal plugin.

**Q**#**19) Which class is used for pagination in Bootstrap?**

**Answer:** To add pagination on the webpage we have to use the class **.pagination**.

**Q**#**20) Explain what is Bootstrap collapsing elements.**

**Answer:** It allows you to collapse any particular element without using any JavaScript code.

To use this feature in bootstrap you have to add data-toggle=” collapse” to the controller element along with a data target to automatically assign the control of a collapsible element. We can use this by writing **.collapse(options)**etc.

**Q**#**21) What is Bootstrap Well?**

**Answer:** Bootstrap well is a form of container which thrives or makes the content to look recessed on the web page. It also wraps the content by using **.well**class.

**Q**#**22) Explain the uses of carousel plugin in Bootstrap.**

**Answer:** Carousel plugin in bootstrap is used to make sliders in the web pages or your site. There are several carousel plugins that are used in bootstrap to display large contents within a small space by adding sliders.

**What are the key components of Bootstrap?**

The key components of Bootstrap are

* CSS : It comes with plenty of CSS files
* Scaffolding : It provides a basic structure with Grid system , link styles and background
* Layout Components : List of layout components
* JavaScript Plugins: It contains many jQuery and JavaScript plugins
* Customize: To get your own version of framework you can customize your components

**What are offset columns in Bootstrap?**

For more specialized layouts offsets are a useful feature. For more spacing they can be used by pushing column over.

**8) What is column ordering in Bootstrap?**

Column ordering is one of the feature available in bootstrap and you can easily write columns in an order and show them in another one. With **.col-md-push-\*** and **.col-md-pull-\***

the order of the column can be easily changed.

**11) What is the use of Jumbotron in Bootstrap?**

In bootstrap, Jumbotron is generally used for content that you want to highlight like some slogan or marketing headline etc. in other words it is used to enlarge the size of the headings and to add a margin for landing page content

To use the Jumbotron in Bootstrap

* Create a container <div> with the class of .jumbotron

**20) Explain what media object in Bootstrap is and what are their types?**

Media objects in Bootstrap enables to put media object like image, video or audio to the left or right of the content blocks. Media element can be created using the class **.media** and the source is specified in using the class **.media-object.**Media-objects are of two types,

They are of two types

* .media
* .media-list

**Q**: **What do you mean by Glyphicons? How do you use them?**  
**A**: Glyphicons are icon fonts that are used in web projects. Although Glyphicons Halflings aren’t free and require licensing in general, they are available free of cost for Bootstrap projects. Add the following code anywhere you wish to use the Glyphicons:

<span **class** = “glyphicon glyphicon-search”></**span**>

**Can you explain Bootstrap breadcrumb?**  
A: A Bootstrap breadcrumb is a great way to display hierarchy-based information for a website. Simply, it is an unordered list with a .breadcrumb class. CSS automatically adds the separator for a Bootstrap breadcrumb.

In blogs, the breadcrumb can display categories, publishing dates, or tags. It indicates the present page’s location within a navigational hierarchy.

Responsive breakpoints

// Extra small devices (portrait phones, less than 576px)

@media (max-width: 575.98px) { ... }

// Small devices (landscape phones, 576px and up)

@media (min-width: 576px) and (max-width: 767.98px) { ... }

// Medium devices (tablets, 768px and up)

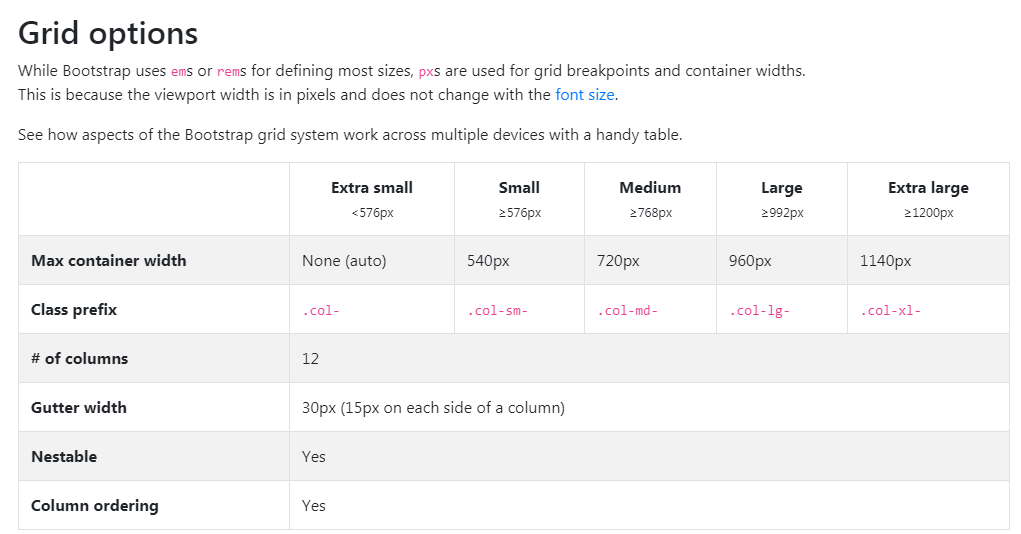
@media (min-width: 768px) and (max-width: 991.98px) { ... }

// Large devices (desktops, 992px and up)

@media (min-width: 992px) and (max-width: 1199.98px) { ... }

// Extra large devices (large desktops, 1200px and up)

@media (min-width: 1200px) { ... }



**CSS:**

### 1) What is CSS?

CSS stands for Cascading Style Sheet. It is a popular styling language which is used with HTML to design websites. It can also be used with any XML documents including plain XML, SVG, and XUL.[More details..](https://www.javatpoint.com/what-is-css)

### 3) What are the different variations of CSS?

Following are the different variations of CSS:

* CSS1
* CSS2
* CSS2.1
* CSS3
* CSS4

### 4) How can you integrate CSS on a web page?

There are three methods to integrate CSS on web pages.

1. Inline method - It is used to insert style sheets in HTML document
2. Embedded/Internal method - It is used to add a unique style to a single document
3. Linked/Imported/External method - It is used when you want to make changes on multiple pages.

### 7) What are the CSS frameworks?

CSS frameworks are the preplanned libraries which make easy and more standard compliant web page styling. The frequently used CSS frameworks are: -

* Bootstrap
* Foundation
* Semantic UI
* Gumby
* Ulkit

### 11) What is a CSS selector?

It is a string that identifies the elements to which a particular declaration apply. It is also referred as a link between the HTML document and the style sheet. It is equivalent of HTML elements. There are several different types of selectors in CSS: -

* CSS Element Selector
* CSS Id Selector
* CSS Class Selector
* CSS Universal Selector
* CSS Group Selector

### 19) Name the property for controlling the image position in the background.

The background-position property is used to define the initial position of the background image. By default, the background image is placed on the top-left of the webpage.

You can set the following positions:

1. center
2. top
3. bottom
4. left
5. right

### 20) Name the property for controlling the image scroll in the background.

The background-attachment property is used to specify if the background image is fixed or scroll with the rest of the page in the browser window. If you set fixed the background image, then the image not move during scrolling in the browser. Let's take an example with the fixed background image.

1. background: white url('bbb.gif');
2. background-repeat: no-repeat;
3. background-attachment: fixed;

### 23) What are the advantages of External Style Sheets?

* You can create classes for reusing it in many documents.
* By using it, you can control the styles of multiple documents from one file.
* In complex situations, you can use selectors and grouping methods to apply styles.

### 25) What is RWD?

RWD stands for Responsive Web Design. This technique is used to display the designed page perfectly on every screen size and device, for example, mobile, tablet, desktop and laptop. You don't need to create a different page for each device.

### 26) What are the benefits of CSS sprites?

If a web page has a large number of images that take a longer time to load because each image separately sends out an HTTP request. The concept of CSS sprites is used to reduce the loading time for a web page because it combines the various small images into one image. It reduces the number of HTTP requests and hence the loading time.

### 31) What is the purpose of the z-index and how is it used?

The z-index helps to specify the stack order of positioned elements that may overlap one another. The z-index default value is zero and can take on either a positive or negative number.

An element with a higher z-index is always stacked above than a lower index.

Z-Index can take the following values:

* **Auto:** Sets the stack order equal to its parents.
* **Number:** Orders the stack order.
* **Initial:** Sets this property to its default value (0).
* **Inherit:** Inherits this property from its parent element.

### 32) Explain the difference between visibility: hidden and display: none?

**visibility: hidden** hides the element, but it occupies space and affects the layout of the document.

**display: none** also hides the element but not occupy space. It will not affect the layout of the document.

### 33) What do you understand by W3C?

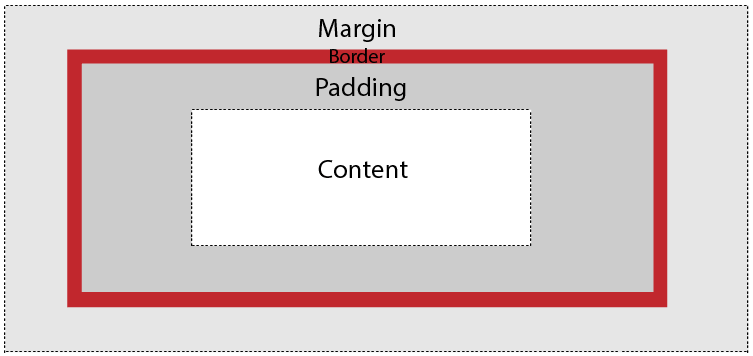
W3C stands for World Wide Web Consortium. Its purpose is to deliver the information of the World Wide Web. It also develops rules and guidelines for the Web.

### 34) What is the CSS Box model and what are its elements?

The CSS box model is used to define the design and layout of elements of CSS.

The elements are:

* Margin - It removes the area around the border. It is transparent.
* Border - It represents the area around the padding
* Padding - It removes the area around the content. It is transparent.
* Content - It represents the content like text, images, etc.



### 35) What is the difference between CSS2 and CSS3?

The main difference between CSS2 and CSS3 is that CSS3 is divided into different sections which are also known as modules. Unlike CSS2, CSS3 modules are supported by many browsers.

Apart from that, CSS3 contains new General Sibling Combinators which is responsible for matching the sibling elements with the given elements.

**Q #8) Explain the concept of Tweening.**

**Answer:**Tweening is the process in which we create intermediate frames between two images to get the appearance of the first image which develops into the second image.

It is mainly used for creating animation.

**Q #2) Name all the modules which are used in the current version of CSS.**

**Answer: There are several modules in CSS as stated below:**

* Selectors
* Box Model
* Backgrounds and Borders
* Text Effects
* 2D/3D Transformations
* Animations
* Multiple Column Layout
* User Interface.

**Q #15) How will you add border images to an HTML element?**

**Answer:**We can set the image to be used as the border-image alongside an element by using the property of CSS “border-image”.

**Example:**

#borderimg {

    border: 15px solid transparent;

    padding: 20px;

    border-image: url(border.png) 30 round;

}

**Q #16) What are gradients in CSS?**

**Answer:**It is a property of CSS which allows you to display a smooth transformation between two or more than two specified colors.

There are two types of gradients that are present in CSS. They are:

* Linear gradient
* Radial Gradient

**Q #17) What is CSS flexbox?**

**Answer:**It allows you to design a flexible responsive layout structure without using any float or positioning property of CSS. To use CSS flexbox you need to define a flex container initially.

**Example:**

<!DOCTYPE html>

<html>

<head>

<style>

.flex-container {

  display: flex;

  background-color: #f4b042;

}

.flex-container > div {

  background-color: #d60a33;

  margin: 10px;

  padding: 20px;

  font-size: 30px;

}

</style>

</head><body>

<div class="flex-container">

  <div>1</div>

  <div>2</div>

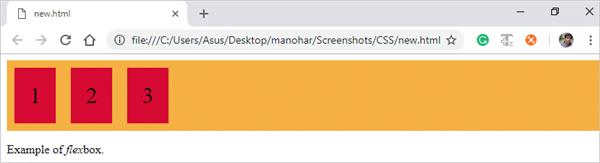
  <div>3</div>

</div>

<p> Example of  <em>flex</em>box.</p>

</body>

</html>

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/10/flexbox.png)

**Q #18) Write all the properties of the flexbox.**

**Answer:** There are several properties of the flexbox that are used in the HTML webpage.

**They are:**

* flex-direction
* flex-wrap
* flex-flow
* justify-content
* align-items
* align-content

**Q #23) What is a CSS pseudo-class?**

**Answer:**It is a class that is used to define a special state of an HTML element.

This class can be used by styling an element when a user snooped over it and also it can style an HTML element when it gets the focus.

selector:pseudo-class {

property:value;

}

**Q #25) What is CSS opacity?**

**Answer:**It is the property that elaborates on the transparency of an element.

By this property, we can transparent the image that can take the values from 0.0-1.0, if the value is lower then the image is more transparent. IE8 and earlier versions of the browser can take the values from 0-100.

img {

opacity: 0.6;

filter: alpha(opacity=60); /\* For IE8 and earlier \*/}

**Q #28) What are the differences between relative and absolute in CSS?**

**Answer:**The main difference between relative and absolute is that “relative” is used for the same tag in CSS and it means that if we write the left:10px then the padding will shift to 10px in the left while absolute is totally relative to the non-static parent.

It means if we write left:10px then the result will be 10px far from the left edge of the parent element.

**Q #31) Differentiate between inline and block element.**

**Answer:**Inline element does not have an element to set width and height and also it does not have the line break.

**Example:** em, strong,span etc.

**Block element specification:**

* They do have the line break.
* They define the width by setting a container and also allow setting height.
* It can also contain an element that occurs in the inline element.

**Example:**

width and height  
max-width and max-height  
min-width and min-height  
hi (i=1-6)- heading element  
p- Paragraph element.

## React JS:

## Babel

## Parcel Bundler

## React

## React DOM

## Live server to identify changes

## Bundlers and transfiler

## Life cycle of React

## Fetch()

## Stateless Functional Components

## React Router

## Heigher-Order Components

## User based browser Events

## Local Storage

## Life cycle of React

## componentDidMount()

## componentWillUnmount()

## SSE – Server Sent Events

import **{** Injectable, NgZone **}** from "@angular/core";

import **{** Observable **}** from "rxjs";

@Injectable**({**

providedIn: "root"

**})**

export **class** SseService **{**

constructor**(**private \_zone: NgZone**)** **{}**

getServerSentEvent**(**url: string**)**: Observable**<**any**>** **{**

**return** Observable.create**(**observer =**>** **{**

const eventSource = **this**.getEventSource**(**url**)**;

eventSource.onmessage = event =**>** **{**

**this**.\_zone.run**(()** =**>** **{**

observer.next**(**event**)**;

**})**;

**}**;

eventSource.onerror = error =**>** **{**

**this**.\_zone.run**(()** =**>** **{**

observer.error**(**error**)**;

**})**;

**}**;

**})**;

**}**

private getEventSource**(**url: string**)**: EventSource **{**

**return** **new** EventSource**(**url**)**;

**}**

**}**

## <https://github.com/lydiahallie/javascript-questions/>

## <https://www.w3resource.com/javascript-exercises/>

## Write a JavaScript program to create a Clock. Note: The output will come every second. Expected Console Output: "14:37:42"

## Answer:

## function my\_Clock()

## {

## this.cur\_date = new Date();

## this.hours = this.cur\_date.getHours();

## this.minutes = this.cur\_date.getMinutes();

## this.seconds = this.cur\_date.getSeconds();

## }

## my\_Clock.prototype.run = function ()

## {

## setInterval(this.update.bind(this), 1000);

## };

## my\_Clock.prototype.update = function ()

## {

## this.updateTime(1);

## console.log(this.hours + ":" + this.minutes + ":" + this.seconds);

## };

## my\_Clock.prototype.updateTime = function (secs)

## {

## this.seconds+= secs;

## if (this.seconds >= 60)

## {

## this.minutes++;

## this.seconds= 0;

## }

## if (this.minutes >= 60)

## {

## this.hours++;

## this.minutes=0;

## }

## if (this.hours >= 24)

## {

## this.hours = 0;

## }

## };

## var clock = new my\_Clock();

## clock.run();

## sum (1)(2)(3)(4)();

## let sum = a => b => return b ? sum (a + b) ? b;

## DE bouncing and Throttling

## DE bouncing: It will return a method after some particular interval of time

## 

## 

## Event Delegation

## 

## 

## Call Bind Apply

## Polly fill for bind

## 

## Event capturing and bubbling

## 

## Currying In JAvascript

## 

## Breaking records Hacker rank question

## 

## 

## 

## 

## Strip key from Object

## 

## 

## 

## 

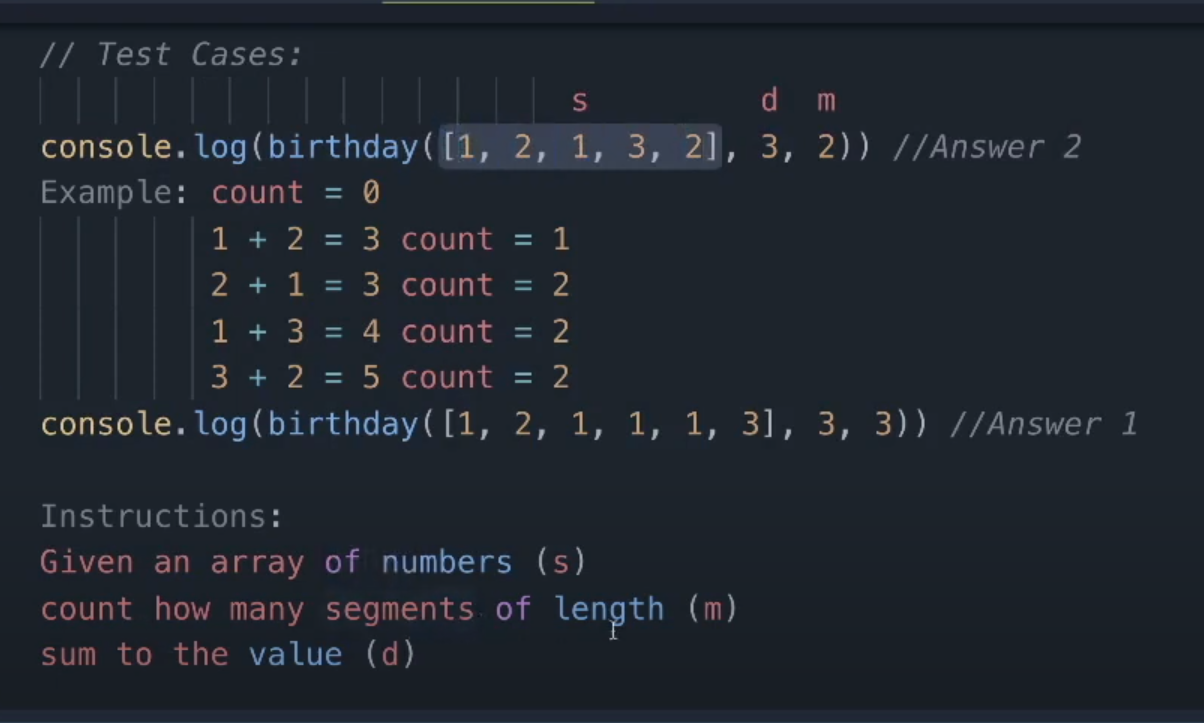
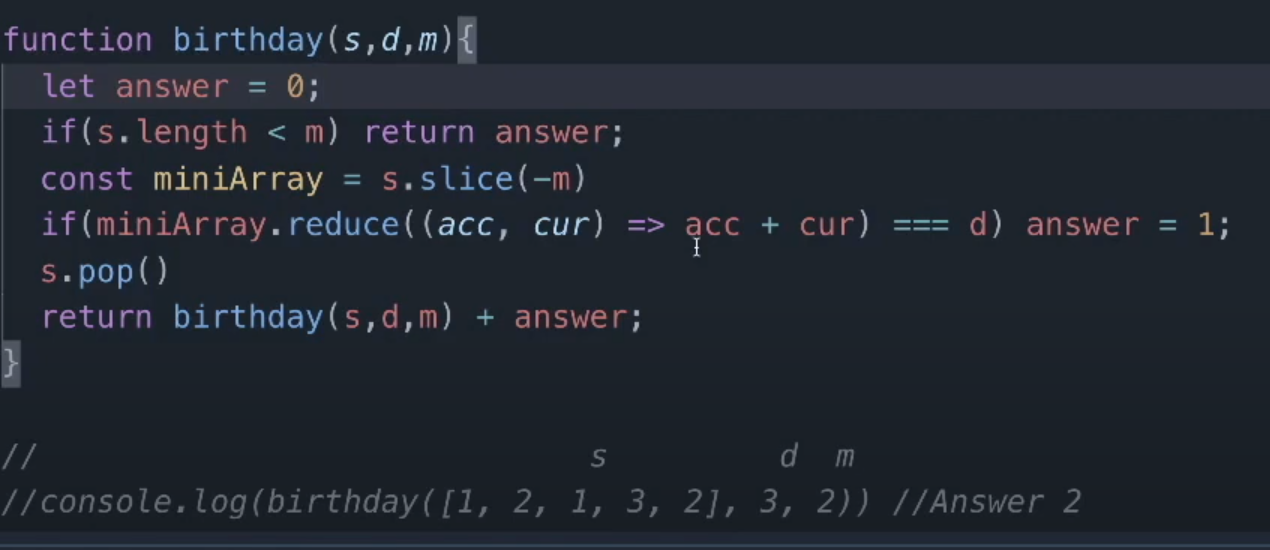
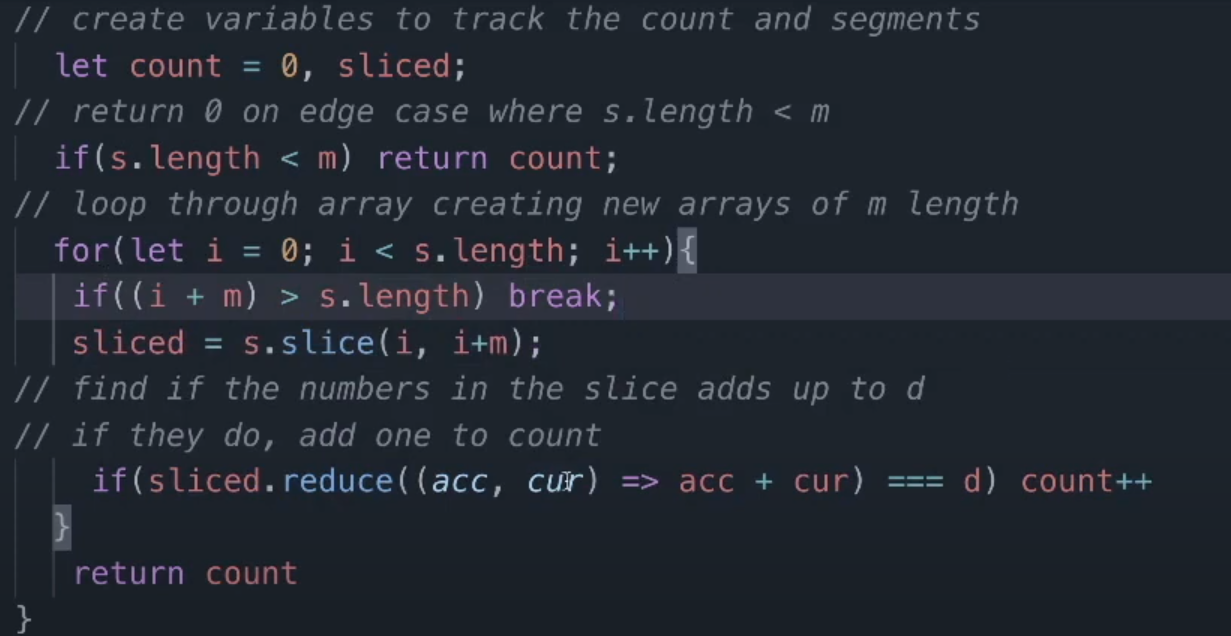
## Diagonal Difference of Matrix

## 

## Grading Students

## 

# Birthday Chocolate from HackerRank



# Sock Merchant problem in JavaScript

## 

## Design Patterns

## Creation patterns (Factory, Singleton)

## Structural Patterns

## Behavioral Patterns

## Strategy Pattern

## Iterator Pattern

## Observer Pattern

## Proxy Pattern

## Mediator Pattern

## Npm install subsink

## At @ngrx/data

## Improve performance in angular application

* Remove the unnecessary change detection that slows down the application.
* Add OnPush at required places.
* Slow HTTP requests(use a service aggregator like GraphQL)
* Optimize the hosting (through static cache content, using PWA)
* Remove unnecessary mathematical recomputations
* Reduce the size of bootstrap logic

# ChangeDetectionStrategy.OnPush

# Detaching the Change Detector

# Local Change Detection

# Run outside Angular

# Use pure pipes

# Use trackBy option for \*ngFor directive

# Optimize template expressions

# Web Workers

# Lazy-Loading

# Preloading

<https://medium.com/@caymanbruce/finding-duplicate-characters-in-a-string-in-javascript-94e2cb23ab5e>

const str = "afewreeeeeeociwddwjejjj";

function findRepeat(str) {

const arr = str.split('');

const hash = new Map();

const result = [];

const nonRepeatedResult = [];

// If repeat the value is false, if no repeat the value is true

for (let i = 0; i < arr.length; i++) {

if (hash.get(arr[i]) === undefined) {

hash.set(arr[i], { exists:true, count: 1});

} else {

const value = hash.get(arr[i]);

if (value.exists) {

hash.set(arr[i], {exists:value.exists, count: ++value.count});

}

}

}

hash.forEach((v, k) => {

if (v.count == 1) {

nonRepeatedResult.push({value:k, count:v.count});

} else if (v.count > 1) {

result.push({value:k, count:v.count});

}

});

result.sort(function (a, b) {

return b.count - a.count

})

return [result[0]];

}

console.log(...findRepeat(str));

https://dev.to/sarah\_chima/anagrams-checker-three-javascript-solutions-53kp

# Anagrams Checker - Three JavaScript Solutions

**Solution - 1**fu

The runtime complexity of a for loop is linear i.e O(n). In this case, there are 3 consecutive forloops which are not nested. Ignoring constants and other factors, the time complexity is approximately linear i.e. O(n).n

function anagrams(stringA, stringB) {

/\*First, we remove any non-alphabet character using regex and convert

convert the strings to lowercase. \*/

stringA = stringA.replace(/[^\w]/g, "").toLowerCase()

stringB = stringB.replace(/[^\w]/g, "").toLowerCase()

//Get the character map of both strings

const charMapA = getCharMap(stringA)

const charMapB = getCharMap(stringB)

/\* Next, we loop through each character in the charMapA,

and check if it exists in charMapB and has the same value as

in charMapA. If it does not, return false \*/

for (let char in charMapA) {

if (charMapA[char] !== charMapB[char]) {

return false

}

}

return true

}

function getCharMap(string) {

// We define an empty object that will hold the key - value pairs.

let charMap = {}

/\*We loop through each character in the string. if the character

already exists in the map, increase the value, otherwise add it

to the map with a value of 1 \*/

for (let char of string) {

charMap[char] = charMap[char] + 1 || 1

}

return charMap

}

**Solution 2**

Array.sort uses merge sort so its time complexity is O(nlogn).

function anagrams(stringA, stringB) {

/\*First, we remove any non-alphabet character using regex and convert

convert the strings to lowercase. \*/

stringA = stringA.replace(/[^\w]/g, '').toLowerCase()

stringB = stringB.replace(/[^\w]/g, '').toLowerCase()

return sortString(stringA) === sortString(stringB)

}

/\*This function sorts the strings\*/

function sortString(string) {

return string.split('').sort().join('');

}

**Data Structures**

1. Stack
2. Set
3. Queue
4. Priority Queue
5. Binary Search Tree
6. Binary Search Tree : Traversal & Height
7. Hash Tables
8. Linked List
9. Trie
10. Heap

function vendingMachine(snack, cash) {

const snacks = [

{ name: 'Espresso', price: 1 },

{ name: 'Cappuccino', price: 2.50 },

{ name: 'Chocolate', price: 2 },

{ name: 'Potato', price: 3.50 }

];

const selected\_snack = snacks.find(item => item.name === snack);

if (selected\_snack) {

if (selected\_snack.price === cash) {

return `Your ${snack} have been served`;

} else {

if (selected\_snack.price > cash) {

return `Insufficient funds. Please insert more cash.`;

}

else {

return `Your ${snack} have been served. Here is your $${cash - selected\_snack.price} change.`;

}

}

} else {

return `${snack} does not exist. Please try again`

}

};

console.log(vendingMachine('Espresso', 12));