Javascript:

1. What is your favourite new javascript feature and why?

Ans:- Well If we are talking about the javascript features, There are a lot to say about the javascript family. From the latest(ES8, ES9), I would like to choose Rest Operator and “object.values()”, which work on the javascript object and return the array of the values of that object. For ex ->   
 let obj = { name: ‘William’ , RollNo: 1, subject: ‘history’, marks: 78 }  
 var values = Object.values(obj);

// returns -> [“William”, 1, “history”, 78]

1. Explain an interesting way in which you have used this javascript feature.

Ans:- Rest Operator =>

const students = { Max: 1 , John: 2, Vin: 3, latte: 4};

const { John, ...rest } = students;

console.log(rest) => { Max: 1, Vin: 3, latte: 4}  
So, It copy the rest of the object’s key-value pair which we didn’t mention particularly.

There are so many features I personally like to use in my all projects are ->  
=> Asynchronous functions(using async, wait),   
=> Template Strings(ex -> let name=”Rock”; let sentence= `I am ${name}” ) ,  
=> Arrow functions (ex -> const newFunction = (parameter) => { return parameter; } ) etc.

1. Is there any difference between regular function syntax and the shorter arrow function syntax? (Write the answer in your own words)

Ans:- Well, If we are talking about syntax, The arrow functions are very short to syntax compare to the regular function. There is no need to write return keyword in arrow function, you can just replace that with parentheses. The main difference is that, We can not use “this” keyword inside the arrow func. but in regular func. we usually do. And In terms of loading, the arrow func. load whenever they are called (like lazy loading) but the regular func. loads initially.

1. What is the difference between ‘myFunctionCall(++foo)’ and ‘myFunctionCall(foo++)’

Ans:- In short, whenever we use ‘++’ before, the javascript execute it like -> foo = foo+1;

So, first It increment the real value of foo and then use that incremented value throughout the method. And in case of ‘foo++’, It will use the actual value inside the method and then increment the actual value by 1.

1. In your own words, explain what a javascript ‘class’ is and how it differs from a function.

Ans:- The javascript class are the special version of javascript functions with some extra functionality in it like constructor. Classes are just a blueprint for the objects like the state of class. So, the state is carried by instances (by creating the class object) and the methods are carried by classes. Though the classes and functions are same in javascript, the main difference are in their syntax.

Css:

1. In your own words, explain css specificity.

Ans:- CSS Specificity is the set of the rules applied to CSS selectors in order to determine which style is applied to an element. The more specific a CSS style is, the higher point value it accrues, and the likelier it is to be present on the element's style.

1. In your own words, explain, what is ‘!important’ in css. Also how does it work? Are there any special circumstances when using it, where it’s behaviour might not be what you expect?

Ans:- The !important property in CSS is used to provide more weight (importance) than normal property. In CSS, the !important means that “this is important”, ignore all the subsequent rules, and apply !important rule and the !important keyword must be placed at the end of the line, immediately before the semicolon.

1. What is your prefered layout system: inline-block, floating + clearing, flex, grid, other? And why?

Ans:- I prefered all these layout systems. Mostky I prefered Flex.

flexbox is truly the first CSS layout tool. These is a few benefits of flex:

1 Equal column height

2 One-fifth grid

3 Change element order

4 Easy to align DOM elements in vertically and horizontally

There are lots of usefull properties in flex

1. Are negative margins legal and what do they do (margin: -20px)?

Ans:- Yes,negative margins are legal. Because W3C even says that Negative values for margin properties are allowed.

An element's margin controls the amount of space between an element's border and surrounding elements. If you set an element's margin to a negative value, the element will grow larger.

By using this code (margin: -20px). We show the result this Element moves in top/left direction, Succeeding element moves in bottom and right direction overlapping it.

1. If a <div/> has no margin or other styling and a <p/> tag inside of it has a margin top of some kind, the margin from the <p/> tag will show up on the div instead (the margin will show above the div not inside of it), why is this? What are the different things that can be done to prevent it?

Ans:- It’s correct behavior for margins. The first child’s top margin will ‘escape’ out the top of the parent, effectively pushing the parent down. There are a couple of ways to prevent it from happening. If the parent has a top border or top padding, the child’s top margin will have something to ‘push’ against and not escape. Or if you change the block formatting context of the parent, it will fully contain the children (with margins). To change the BFC you could float the parent, give it a different positioning context, or add overflow:hidden (any one would work).

Unit tests:

1. What technologies do you use to unit test your react components?

Ans:- We usually use React Testing Library or snapshot Testing to test our React Component.

1. Are there any pitfalls associated with this technology that have caused you difficulty in the past?

Ans:- Initially when I started using React Testing Library, I have faced a lot of issues regarding the pre build methods provided by it.

1. How do you test in your unit tests to see if the correct properties are being passed to child components.

Ans:- While using React Testing Library, To test the states in the child, We often use getByText query method to get the value from array destructuring and display the text value to the UI. So, it will match that value with expected. Then on every change, the props are going to be changed, then we will need to render the parent along with the child component again in order to test the parent property inside the child and again that will test through the text displayed on the UI.

React:

1. React test step1:  
     
   Create a react component that has a <div/> with a border.  
   Inside this <div/> should be a <span/> that displays the ‘live’ width of the browser window at all times. Keep in mind that the size of the window could easily be changed by the user and you should reflect this.

Ans:-

import React, { useState } from "react";

const App = (props) => {

const [width, setWidth] = useState(window.innerWidth);

const [height, setHeight] = useState(window.innerHeight);

window.addEventListener("resize", () => {

setHeight(window.innerHeight);

setWidth(window.innerWidth);

});

return (

<div style={{>

<span> Your current window height is - {height} </span>

<span> Your current window width is - {width} </span>

</div>

)

}

1. React test step2:  
     
   Inside the <div/> you created in the previous step, add a text input that, as a number is entered into it, uses that number to set the height of the div itself in pixels, live as you update the text field (keypress not change event).

Ans:-

import React, { useState } from "react";

const App = (props) => {

const [width, setWidth] = useState(window.innerWidth);

const [height, setHeight] = useState(window.innerHeight);

const [divHeight, setdivHeight] = useState(100);

window.addEventListener("resize", () => {

setHeight(window.innerHeight);

setWidth(window.innerWidth);

});

const divStyle = {

height: divHeight + 'px',

border: '10px solid #409cff',

};

const changeDivHeight = e => (setdivHeight(e.target.value));

return (

<div style={divStyle}>

<span> Your current window height is - {height} </span>

<span> Your current window width is - {width} </span> <br/>

<input

type="number"

value={divHeight}

onChange={e => changeDivHeight(e)} />

</div>

)

}

1. React test step3:  
     
   Add the following code to your project root (same project as in step 2, but add the code in the global / window space):   
     
    Let divHeight;  
    window.setDivHeight = (height) => divHeight = height;  
   Add a HOC for your div component that allows you to set the height of your <div/> component from the previous steps by calling that external function.

Ans:-

//HOC.js file ->

import React, { Component } from 'react';

const Hoc = (HocComponent) => {

return class extends Component {

state = {

height: 100

}

changeDivHeight = (e) => (*this*.setState({ height: e.target.value }))

render() {

return (<div style={{ "height": `${*this*.state.height}px`, border: '10px solid #409cff' }}>

<HocComponent

changeDivHeight={*this*.changeDivHeight}

divHeight={*this*.state.height}

></HocComponent>

</div>

);

}

}

}

export default Hoc;

//App.js file

import React, { useState } from "react";

import Hoc from "./HOC";

const App = (props) => {

const [width, setWidth] = useState(window.innerWidth);

const [height, setHeight] = useState(window.innerHeight);

window.addEventListener("resize", () => {

setHeight(window.innerHeight);

setWidth(window.innerWidth);

});

return (

<>

<span> Your current window height is - {height} </span>

<span> Your current window width is - {width} </span> <br />

<input type="number" value={props.divHeight} onChange={e => props.changeDivHeight(e)} />

</>

)

}

export default Hoc(App);

RXjs:

1. What are the differences between Subject, BehaviorSubject and ReplaySubject? And in what situation would you use each of these (please provide example scenarios)?

Ans:- The Subject doesn’t hold any value so it doesn’t return the current value on subscription. It triggers only on ‘.next(value)’ call and return the value.  
  
whereas the BehaviorSubject holds single value and return the initial value or the current value on subscription.   
  
But the replaySubject is used when you want BehaviorSubject to hold more than one value.

In the Next Subscribers they receive, in case of ->

Subject => Only upcoming values.

BehaviorSubject => One previous value and upcoming values.

ReplaySubject => All previous values and upcoming values.

Examples ->

import \* as Rx from 'rxjs';

const subject1 = new Rx.Subject();

subject1.next(1);

subject1.subscribe(x => console.log(x)); *// will print nothing -> because we subscribed after the emission and it does not hold the value.*

const subject2 = new Rx.Subject();

subject2.subscribe(x => console.log(x)); *// print 1 -> because the emission happened after the subscription.*

subject2.next(1);

const behavSubject1 = new Rx.BehaviorSubject(1);

behavSubject1.next(2);

behavSubject1.subscribe(x => console.log(x)); *// print 2 -> because it holds the value.*

const behavSubject2 = new Rx.BehaviorSubject(1);

behavSubject2.subscribe(x => console.log('val:', x)); *// print 1 -> default value*

behavSubject2.next(2) *// just because of next emission will print 2*

1. If you have an array of values in a stream and you wish to pipe it such that it will emit the arrays values individually, one by one and wait for them all to be completed before processing another array, how would you do this? Please provide a code example.  
   E.g.  
   [1,2] -> [3,4] -> [5,6,7]  
   Should produce:  
   1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7

Ans:- In this scenario, we can simply use map to get the whole array and then apply filter to it, to filter out the expected/required output. See below the code sample-

let sampleArr = [[1, 2], [3, 4], [5, 6, 7]];

let finalArr = [];

sampleArr.map(item => item.filter(value => finalArr = finalArr.concat(value)))

//finalArr = [1, 2, 3, 4, 5, 6, 7];

1. If you have a stream that receives individual values and would like to pipe it such that it builds an array out of these values, emitting the updated array each time a new value is added to it, how would you do this? Please provide a code example.  
   E.g.  
   1 -> 2 -> 3 -> 4  
   Should produce:  
   [1] -> [1,2] -> [1,2,3] -> [1,2,3,4]

Ans:- We can do this using map and spread operator.  
See below the code sample -

let sampleArr = [1, 2, 3, 4];

let tempArr = [];

let finalOutput =sampleArr.map(item => tempArr = [...tempArr,item]);

*//finalOutput = [[1],[1,2],[1,2,3],[1,2,3,4]]*

Twilio:

1. Explain which of the Twilio Api’s you have used. Also explain how and in what scenarios you have used them.

Ans:- I have used the SMS twilio REST API in nodeJs for sending the message to the users from my Twilio Phone number. Initially, we get our token access from our Twilio account and then we can easily send the sms to anyone.  
There will be tree params required(basic) in the pre-build method of twilio i.e. messages. So, the basic params are -   
(1) body (The message sample we wand to send),  
(2) from (our registered twilio number),

(3) to (The client’s number whom we want to send SMS).  
Then simply we get the response either error(if any) or success.