JS Snippets:-

1] finding nth(300th) element out of nth(1099th ) element inside an array ?

= code :

const array1 = [5, 12, 8, 130, 44];

const found = array1.filter((e,index) => index === 2);

console.log(found);

OUTPUT :

[8]

2] How react is faster than angular ?

= ReactJS uses virtual DOM, while **Angular** operates on real DOM and uses change detection to find which component needs updates. While Virtual DOM is considered to be **faster than** real DOM manipulations, the current implementations of change detection in **Angular** make both approaches comparable in terms of performance.

3] What is promise in JavaScript?

= The **Promise** object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value

4] What is Redux used for?

= **Redux** is a predictable state container for JavaScript apps. **Redux** makes it easy to manage the state of your application. Another way of looking at this – it helps you manage the data you display and how you respond to user actions

5] What is redux promise?

= 1. No. That requires **redux**-thunk . To clarify: **redux**-**promise** lets you pass promisesdirectly to dispatch() , or put **promises** inside of an action object.

6] What is route in react JS?

= **React** Router is the standard **routing** library for **React**. From the docs: “**React** Router keeps your UI in sync with the URL. It has a simple API with powerful features like lazy code loading, dynamic **route** matching, and location transition handling built right in.

7] What exactly is the DOM?

= The **DOM** (Document Object Model) is an interface that represents how your HTML and XML documents are read by the browser. It allows a language (JavaScript) to manipulate, structure, and style your website

8] What is the DOM in JavaScript?

= **JavaScript** - Document Object Model or **DOM**. ... A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

9] How is Dom rendered?

= The **DOM** (Document Object Model) is formed from the HTML that is received from a server. Styles are loaded and parsed, forming the CSSOM (CSS Object Model). ... Each of the **rendering** objects contains its corresponding **DOM** object (or a text block) plus the calculated styles.

10] What is DOM manipulation?

= One of the functions mostly used in **DOM** work is: getElementById.**Manipulating**/Changing the **DOM** means using this API to change the document (add elements, remove elements, move elements around etc...). Traversing the **DOM**means navigating it - selecting specific elements, iterating over groups of elements etc...

11] Feature’s of React JS ?

= React JS makes the process of writing components smoother

React JS increases efficiency and makes maintenance easier

React JS provides an ideal solution to high-load application

React ensures stable code

React enhances SEO performance

React JS comes with useful developer tools

12] import React from ‘react’ meaning ?

= Here are the [docs](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import) for import.

import React from 'react'

The above is a default import. Default imports are exported with export default .... There can be only a single default export.

import { Component } from 'react'

But this is a member import (named import). Member imports are exported with export .... There can be many member exports.

13] **Why do we call super? Can we not call it? If we have to call it, what happens if we don’t pass props? Are there any other arguments?.**

**=** In JavaScript, super refers to the parent class constructor. (In our example, it points to the React.Component implementation.)

Importantly, you can’t use this in a constructor until after you’ve called the parent constructor. JavaScript won’t let you:

class Checkbox extends React.Component {

constructor(props) {

// 🔴 Can’t use `this` yet

super(props);

// ✅ Now it’s okay though

this.state = { isOn: true };

}

// ...

}

There’s a good reason for why JavaScript enforces that parent constructor runs before you touch this. Consider a class hierarchy:

class Person {

constructor(name) {

this.name = name;

}

}

class PolitePerson extends Person {

constructor(name) {

this.greetColleagues(); // 🔴 This is disallowed, read below why

super(name);

}

greetColleagues() {

alert('Good morning folks!');

}

}

Imagine using this before super call was allowed. A month later, we might change greetColleagues to include the person’s name in the message:

greetColleagues() {

alert('Good morning folks!');

alert('My name is ' + this.name + ', nice to meet you!');

}

But we forgot that this.greetColleagues() is called before the super() call had a chance to set up this.name. So this.name isn’t even defined yet! As you can see, code like this can be very difficult to think about.

To avoid such pitfalls, **JavaScript enforces that if you want to use this in a constructor, you have to call super first.** Let the parent do its thing! And this limitation applies to React components defined as classes too:

constructor(props) {

super(props);

// ✅ Okay to use `this` now

this.state = { isOn: true };

}

14] why pass props?

= You might think that passing props down to super is necessary so that the base React.Component constructor can initialize this.props

// Inside React

class Component {

constructor(props) {

this.props = props;

// ...

}

}

And that’s not far from truth — indeed, that’s [what it does](https://github.com/facebook/react/blob/1d25aa5787d4e19704c049c3cfa985d3b5190e0d/packages/react/src/ReactBaseClasses.js#L22).

But somehow, even if you call super() without the props argument, you’ll still be able to access this.props in the render and other methods.

So even if you forget to pass props to super(), React would still set them right afterwards. There is a reason for that.

So does this mean you can just write super() instead of super(props)?

**Probably not because it’s still confusing.** Sure, React would later assign this.props after your constructor has run. But this.props would still be undefined between the super call and the end of your constructor:

class Component {

constructor(props) {

this.props = props;

// ...

}

}

// Inside your code

class Button extends React.Component {

constructor(props) {

super(); // 😬 We forgot to pass props

console.log(props); // ✅ {}

console.log(this.props); // 😬 undefined }

// ...

}

It can be even more challenging to debug if this happens in some method that’s called from the constructor. **And that’s why I recommend always passing down super(props), even though it isn’t strictly necessary:**

class Button extends React.Component {

constructor(props) {

super(props); // ✅ We passed props

console.log(props); // ✅ {}

console.log(this.props); // ✅ {}

}

// ...

}

This ensures this.props is set even before the constructor exits.

15] what happens when you call setstate() inside render() method ?

= Calling setState() here makes your component a contender for producing infinite loops.

The render() function should be pure, meaning that it does not modify component state, it returns the same result each time it’s invoked, and it does not directly interact with the browser.

In this case, avoid using setState() here.

16] ref is used to refer an/a element/component returned by ?

= When the **ref** attribute is **used** on an HTML **element**, the **ref** created in the constructor with React. createRef() receives the underlying DOM **element** as its current property. When the **ref** attribute is **used** on a custom class **component**, the**ref** object receives the mounted instance of the **component** as its current .

17] arbitrary inputs of components are called ?

= Props

Conceptually, **components** are like JavaScript functions. They accept **arbitrary inputs** (**called** "props") and return React elements describing what should appear on the screen. props is an object available on this inside a **component** class that gives you access to the values that were passed when a **component** was **called**.

18] What is event preventDefault () in react ?

= What is **preventDefault()** in **React**? **React** uses synthetic **events** to handle **events**from button, input and form elements. A synthetic **event** is a shell around the native DOM **event** with additional information for **React**. Sometimes you have to use**event**.**preventDefault**(); in your application.

19] what does react router do

= **React Router is** the standard **routing** library for **React**. From the docs: “**React Router** keeps your UI in sync with the URL. It has a simple API with powerful features like lazy code loading, dynamic route matching, and location transition handling built right in.

20] What is react Redux provider?

= Overview. The <**Provider** /> makes the **Redux** store available to any nested components that have been wrapped in the connect() function. Since any **React**component in a **React Redux** app can be connected, most applications will render a <**Provider**> at the top level, with the entire app's component tree inside of it.

21] Why should I use react JS?

= It is used to handle all views of an application for any web or mobile applications.**ReactJS** is also used to reuse UI components. **React** enables developers to create web applications that can change your data without reloading your page. The main advantage of **React JS** is that it is scalable, simple and fast.

22] What is a store in react ?

= A **store** is basically just a plain JavaScript object that allows components to share state. In a way, we can think of a **store** as a database. On the most fundamental level, both constructs allow us to **store** data in some form or another.

23] what is public class field syntax ?

**Public Class Fields** allow you to add instance properties to the **class** definition with the assignment operator

24] What is ReactDom ?

ReactDom is a package that provides Dom (Document Object Model) specifies methods that can be used in a web app to enable an efficient way of managing DOM elements that are rendered on the web page.

25] why we use Construtor in class ?

A class is a type of function, but instead of using the keyword function to initiate it, we use the keyword class, and the properties are assigned inside a constructor() method.

26] How do you round the number 7.25, to the nearest integer?

Math.round(7.25)

26] Difference between JSX and HTML ?

1. What is JSX ?

* JSX (JavaScript + XML) is an extension of JavaScript that allows you to write HTML directly within JavaScript, which has a few benefits of making your code more readable
* JSX’s main intention is to be used by transpilers or preprocessors to transform these tokens into standard ECMAScript.
* Since JSX is not a valid JS code, it needs to be compiled into JS with a tool like Babel or similar.
* Since JSX is a syntactic extension of JS, you can write JS directly within JSX. To do so, you’ll need to brace your code within the curly braces for it to be treated as JS:

{ 'insert your JavaScript code here' }

# 27] AJAX - Send a Request To a Server ?

# To send a request to a server, we use the open() and send() methods of

# the XMLHttpRequest object:

# e.g

# xhttp.open("GET", "ajax\_info.txt", true);

# xhttp.send();

|  |  |
| --- | --- |
| **Method** | **Description** |
| open(*method, url, async*) | Specifies the type of request  *method*: the type of request: GET or POST *url*: the server (file) location *async*: true (asynchronous) or false (synchronous) |
| send() | Sends the request to the server (used for GET) |
| send(*string*) | Sends the request to the server (used for POST) |

## GET or POST? :-

GET is simpler and faster than POST, and can be used in most cases.

However, always use POST requests when:

* A cached file is not an option (update a file or database on the server).
* Sending a large amount of data to the server (POST has no size limitations).
* Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

## GET Requests

xhttp.open("GET", "demo\_get.asp", true);  
xhttp.send();

## POST Requests :

xhttp.open("POST", "demo\_post.asp", true);  
xhttp.send();

To POST data like an HTML form, add an HTTP header with setRequestHeader(). Specify the data you want to send in the send() method:

<button type="button" onclick="loadDoc()">Request data</button>

<p id="demo"></p>

<script>

function loadDoc() {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

document.getElementById("demo").innerHTML = this.responseText;

}

};

xhttp.open("POST", "demo\_post2.asp", true);

xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

xhttp.send("fname=Henry&lname=Ford");

}

</script>

28] javascript is synchronous or asynchronous ?

**JavaScript** is always **synchronous** and single-threaded. ... **JavaScript** is only **asynchronous** in the sense that it can make, for example, Ajax calls. The Ajax call will stop executing and other code will be able to execute until the call returns (successfully or otherwise), at which point the callback will run **synchronously**.

# 29] JavaScript Array push() Method ?

The push() method adds new items to the end of an array, and returns the new length.

**Note:** The new item(s) will be added at the end of the array.

**Note:** This method changes the length of the array.

# 30] JavaScript Array sort() Method ?

The sort() method sorts the items of an array.

The sort order can be either alphabetic or numeric, and either ascending (up) or descending (down).

By default, the sort() method sorts the values as strings in alphabetical and ascending order.

This works well for strings ("Apple" comes before "Banana"). However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the sort() method will produce an incorrect result when sorting numbers.

You can fix this by providing a "compare function" (See "Parameter Values" below).

**Note:** This method changes the original array

e.g.

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>

var fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo").innerHTML = fruits;

function myFunction() {

fruits.sort();

document.getElementById("demo").innerHTML = fruits;

}

</script>

# 31] JavaScript Array toString() Method ?

The toString() method returns a string with all the array values, separated by commas.

**Note:** This method will not change the original array.

e.g.

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>

function myFunction() {

var fruits = ["Banana", "Orange", "Apple", "Mango"];

var x = fruits.toString();

document.getElementById("demo").innerHTML = x;

}

</script>

32] Array ProtoType ?

The prototype constructor allows you to add new properties and methods to the Array() object.

When constructing a property, ALL arrays will be given the property, and its value, as default.

When constructing a method, ALL arrays will have this method available.

**Note:** Array.prototype does not refer to a single array, but to the Array() object itself.

**Note:** Prototype is a global object constructor which is available for all JavaScript objects.

e.g.

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>

Array.prototype.myUcase = function() {

var i;

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

this[i] = this[i].toUpperCase();

}

};

function myFunction() {

var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.myUcase();

document.getElementById("demo").innerHTML = fruits;

}

</script>

# 33] JavaScript Array reverse() Method ?

The reverse() method reverses the order of the elements in an array.

**Note:** this method will change the original array.

e.g.

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>

var fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo").innerHTML = fruits;

function myFunction() {

fruits.reverse();

document.getElementById("demo").innerHTML = fruits;

}

</script>

### 34] Difference between local storage and session storage ?

**Local storage and Session storage** are the web srorage objects. **Session storage** is destroyed once the user closes the browser whereas, **Local storage** stores data with no expiration date. The **sessionStorage** object is equal to the **localStorage** object, except that it stores the data for only one **session**.

### 35] How React works?

Below is the sequence of steps which gives an idea about how does react work

* Firstly the react runs the diffing algorithm to identify the changes that are made in the virtual DOM.
* Next step is reconciliation, this is used to update the DOM as per the new features.
* Now, the virtual DOM, which is lightweight in nature and is detached from the specific implementation of the browser.
* Followingly the ReactElements which are present in virtual DOM are used to build basic nodes.
* Finally, if the ReactComponent changes the state; the diffing algorithm runs faster and identify the changes. After identification, it automatically updates the DOM with the change difference.

### 36] **How is ReactJs different from AngularJS?**

The first difference between both of them is their code dependency. ReactJS depends less on the code whereas AngularJS needs a lot of coding to be done. The packaging on React is quite strong as compared to the AngularJS. Another difference is React is equipped with Virtual Dom while the Angular has a Regular DOM. ReactJS is all about the components whereas AngularJS focus mainly on the Models, View as well as on Controllers. AngularJS was developed by Google while the ReactJS is the outcome of facebook. These are some of the common differences between the two.

### **37] What are the differences between the Class component and Functional component ?**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Class Component** | **Functional Component** |
| Syntax | This component requests you to extend from React. Component to create render function that in turn returns a react element | It is just a plain JavaScript function that accepts props as their arguments and returns the react element. |
| Life cycle hooks | Lifecycle hooks are created from the React Component. This class component makes lifecycle hooks available in it. | We cannot use lifecycle hooks in a functional component. |
| Readability | They are very difficult to test and read | They are much easier to test and read |

# 38] What is Ploymorphism ?

# The polymorphism is a core concept of an object-oriented paradigm that provides a way to perform a single action in different forms. It provides an ability to call the same method on different JavaScript objects.

# e.g.

# class A {

# display() {

# console.log("A is invoked")

# }

# }

# class B extends A {

# }

# var b=new B();

# b.sdisplay();

29] Events in JavaScript ?

* Keyboard Event : -

1. [keydown](https://www.w3schools.com/jsref/event_onkeydown.asp) :- The event occurs when the user is pressing a key

e.g

<input type="text" onkeydown="myFunction()">

<script>

function myFunction() {

alert("You pressed a key inside the input field");

}

</script>

1. [keypress](https://www.w3schools.com/jsref/event_onkeypress.asp) :- The event occurs when the user presses a key

e.g

<input type="text" onkeypress="myFunction()">

<script>

function myFunction() {

alert("You pressed a key inside the input field");

}

</script>

1. [keyup](https://www.w3schools.com/jsref/event_onkeyup.asp) :- The event occurs when the user releases a key

e.g

<input type="text" onkeyup="myFunction()">

<script>

function myFunction() {

alert("You pressed a key inside the input field");

}

</script>

* Mouse Event :-

1. [mousedown](https://www.w3schools.com/jsref/event_onmousedown.asp) :- The event occurs when the user presses a mouse button over an

element

e.g.

<p id="myP" onmousedown="mouseDown()" onmouseup="mouseUp()">

Click the text! The mouseDown() function is triggered when the mouse button is pressed down over this paragraph, and sets the color of the text to red. The mouseUp() function is triggered when the mouse button is released, and sets the color of the text to green.

</p>

<script>

function mouseDown() {

document.getElementById("myP").style.color = "red";

}

function mouseUp() {

document.getElementById("myP").style.color = "green";

}

</script>

1. [mouseenter](https://www.w3schools.com/jsref/event_onmouseenter.asp) : - The event occurs when the pointer is moved onto an element

e.g

<img onmouseenter="bigImg(this)" onmouseleave="normalImg(this)" border="0" src="smiley.gif" alt="Smiley" width="32" height="32">

<p>The function bigImg() is triggered when the user moves the mouse pointer onto the image.</p>

<p>The function normalImg() is triggered when the mouse pointer is moved out of the image.</p>

<script>

function bigImg(x) {

x.style.height = "64px";

x.style.width = "64px";

}

function normalImg(x) {

x.style.height = "32px";

x.style.width = "32px";

}

</script>

1. [mouseleave](https://www.w3schools.com/jsref/event_onmouseleave.asp) :- The event occurs when the pointer is moved out of an element

e.g.

<img onmouseenter="bigImg(this)" onmouseleave="normalImg(this)" border="0" src="smiley.gif" alt="Smiley" width="32" height="32">

<p>The function bigImg() is triggered when the user moves the mouse pointer onto the image.</p>

<p>The function normalImg() is triggered when the mouse pointer is moved out of the image.</p>

<script>

function bigImg(x) {

x.style.height = "64px";

x.style.width = "64px";

}

function normalImg(x) {

x.style.height = "32px";

x.style.width = "32px";

}

</script>

1. [mousemove](https://www.w3schools.com/jsref/event_onmousemove.asp) :- The event occurs when the pointer is moving while it is over an element

e.g.

<style>

div {

width: 200px;

height: 100px;

border: 1px solid black;

}

</style>

<body>

<div onmousemove="myFunction(event)" onmouseout="clearCoor()"></div>

<p>Mouse over the rectangle above, and get the coordinates of your mouse pointer.</p>

<p>When the mouse is moved over the div, the p element will display the horizontal and vertical coordinates of your mouse pointer, whose values are returned from the clientX and clientY properties on the

MouseEvent object.</p>

<p id="demo"></p>

<script>

function myFunction(e) {

var x = e.clientX;

var y = e.clientY;

var coor = "Coordinates: (" + x + "," + y + ")";

document.getElementById("demo").innerHTML = coor;

}

function clearCoor() {

document.getElementById("demo").innerHTML = "";

}

</script>

1. [onmouseout](https://www.w3schools.com/jsref/event_onmouseout.asp) :- The event occurs when a user moves the mouse pointer out of an element, or out of one of its children

e.g

<img onmouseover="bigImg(this)" onmouseout="normalImg(this)" border="0" src="smiley.gif" alt="Smiley" width="32" height="32">

<p>The function bigImg() is triggered when the user moves the mouse pointer over the image.</p>

<p>The function normalImg() is triggered when the mouse pointer is moved out of the image.</p>

<script>

function bigImg(x) {

x.style.height = "64px";

x.style.width = "64px";

}

function normalImg(x) {

x.style.height = "32px";

x.style.width = "32px";

}

</script>

1. [onmouseover](https://www.w3schools.com/jsref/event_onmouseover.asp) : - The event occurs when the pointer is moved onto an element, or onto one of its children

e.g.

<img onmouseover="bigImg(this)" onmouseout="normalImg(this)" border="0" src="smiley.gif" alt="Smiley" width="32" height="32">

<p>The function bigImg() is triggered when the user moves the mouse pointer over the image.</p>

<p>The function normalImg() is triggered when the mouse pointer is moved out of the image.</p>

<script>

function bigImg(x) {

x.style.height = "64px";

x.style.width = "64px";

}

function normalImg(x) {

x.style.height = "32px";

x.style.width = "32px";

}

</script>

1. [onmouseup](https://www.w3schools.com/jsref/event_onmouseup.asp) :- The event occurs when a user releases a mouse button over an element

e.g.

<p id="myP" onmousedown="mouseDown()" onmouseup="mouseUp()">

Click the text! The mouseDown() function is triggered when the mouse button is pressed down over this paragraph, and sets the color of the text to red. The mouseUp() function is triggered when the mouse button is released, and sets the color of the text to green.

</p>

<script>

function mouseDown() {

document.getElementById("myP").style.color = "red";

}

function mouseUp() {

document.getElementById("myP").style.color = "green";

}

</script>

1. [onclick](https://www.w3schools.com/jsref/event_onclick.asp) : - The event occurs when the user clicks on an element

e.g

<h1>The onclick Event</h1>

<p>The onclick event is used to trigger a function when an element is clicked on.</p>

<p>Click the button to trigger a function that will output "Hello World" in a p element with id="demo".</p>

<button onclick="myFunction()">Click me</button>

<p id="demo"></p>

<script>

function myFunction() {

document.getElementById("demo").innerHTML = "Hello World";

}

</script>

1. [oncontextmenu](https://www.w3schools.com/jsref/event_oncontextmenu.asp) :- The event occurs when the user right-clicks on an element to open a context menu

e.g

<div oncontextmenu="myFunction()" contextmenu="mymenu">

<p>Right-click inside this box to see the context menu!

<menu type="context" id="mymenu">

<menuitem label="Refresh" onclick="window.location.reload();" icon="ico\_reload.png"></menuitem>

<menu label="Share on...">

<menuitem label="Twitter" icon="ico\_twitter.png" onclick="window.open('//twitter.com/intent/tweet?text=' + window.location.href);"></menuitem>

<menuitem label="Facebook" icon="ico\_facebook.png" onclick="window.open('//facebook.com/sharer/sharer.php?u=' + window.location.href);"></menuitem>

</menu>

<menuitem label="Email This Page" onclick="window.location='mailto:?body='+window.location.href;"></menuitem>

</menu>

</div>

<script>

function myFunction() {

alert("You right-clicked inside the div!");

}

</script>

1. [ondblclick](https://www.w3schools.com/jsref/event_ondblclick.asp) :- The event occurs when the user double-clicks on an element

e.g

<p ondblclick="myFunction()">Double-click this paragraph to trigger a function.</p>

<p id="demo"></p>

<script>

function myFunction() {

document.getElementById("demo").innerHTML = "Hello World";

}

</script>