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.