+16092356405

Javascript.info //great site

Notes

jsv9000.app for event loop

**Infinite currying:**

1. Question: add(1)(2)(4)…(n)();

Ans: let total=0;

function add(i){

  if(i){

    total = total +i;

    return function (j){

      return add(j)

    };

  } else console.log(total);

}

const hello = add(6)(5)(2)();

Ans 2: *function* add(*a*) {

  return *function* (*b*) {

    if (*b*) {

      return add(*a* + *b*);

    }

    return *a*;

  };

}

For below refer add1 function:

console.log(+add1(1));

console.log(+add1(1)(3)(52)(342)); // + operatore will invoke the valueof function. Alternatively can be used Number()

const add1 = (a) =>{

const sum =(y) =>{

if(y===undefined){

return a;

} else {

return add1(a+y);

}

}

sum.valueOf = ()=>{

return a;

}

return sum;

}

Q.2 Compose (evaluate functions from right to left) . It is compose function. Pipe function evaluates from left to right

*const* compose = (...*funcs*) *=>* {

  return *function* (*i*) {

    return *funcs*.reduceRight((*acc*, *fn*) *=>* {

      return fn(*acc*);

    }, *i*);

  };

};

console.log(compose(subtactTwo, multiplyFour)(8));

Q.3 Debouncing:

<input onkeyup="trial()" />

*let* counter = 0;

*const* cb = () *=>* {

  console.log(counter++);

};

*const* debounce = (*fn*, *time*) *=>* {

*let* timer;

  return *function* () {

    if (timer) clearTimeout(timer);

    timer = setTimeout(() *=>* {

      fn();

    }, *time*);

  };

};

*const* trial = debounce(cb, 1000);

Q.4 Throttling (automatic function call after certain time period)

*function* throttle(*fn*, *time*) {

*let* timer = true;

*const* context = this;

  return *function* (*i*) {

    if (timer) {

      timer = false;

      setTimeout(() *=>* {

        fn(*i*);

        timer = true;

      }, *time*);

    }

  };

}

*const* trialthrottle = throttle(cb, 500);

Q.5 How to center Div

Method 1

.box {

*width*: fit-content;

*position*: absolute;

*left*: 50%;

*top*: 50%;

*border*: 2px solid black;

*transform*: translate(-50%, -50%);

}

Method 2

body {

*font-size*: medium;

*font-family*: sans-serif;

*box-sizing*: border-box;

*padding*: 0;

*margin*: 0;

*color*: brown;

*width*: 100%;

*height*: 100vh;

*display*: flex;

*align-items*: center;

*justify-content*: center;

}

.box {

*width*: fit-content;

*border*: 2px solid black;

}

Method 3: Change display type to grid in above example

Method 4: change property of parent element to flex/gride & make the margin auto in child

Q.6 closure?

- if a function is generated inside a parent function then it will have access to the parent function variables (lexical environment) complete environment even if the function is called outside the scope

- If parent function is also nested inside a closure function then again it will have access to that environment.

- closure variable will have higher priority over scope chain

*function* nest1() {

*let* a = 10;

*let* b = 11;

  return *function* nest2() {

    a = 12;

    return *function* inner() {

      console.log(a, b); //it will print a=12 & b=11

    };

  };

}

nest1()()();

disadvantages: overconsumption of memory due to high memory consumption of closure as variables are not garbage collected.

like unnecessary variables in the closures accumulate the place in the memory.

But smart garbage collection in today’s V8 engine of chrome, such garbage collection is done smartly and unnecessary variables present are removed from memory .

Q.7 Memoizing? & encaching

Q.8 data hiding & encapsulation:

Variable can be protected using closures. In example below, count can not be accessed from outside

*function* counter() {

*let* count = 0;

  this.counterIncreament = () *=>* {

    count++;

    console.log(count);

  };

  this.counterDecrement = () *=>* {

    count--;

    console.log(count);

  };

}

*const* finalCounter = new counter();

finalCounter.counterIncreament();

Q.8 Call, bind, apply

*const* obj1 = {

  a: 5,

  b: 10,

  add(*text1*, *text2*) {

    this.total = this.a + this.b;

    return this.total + *text1* + *text2*;

  },

};

*const* obj2 = {

  a: 13,

  b: 15,

};

console.log(obj1.add.call(obj2, "h", "y")); //28hy

console.log(obj1.add.apply(obj2, ["h", "y"])); // 28hy;

console.log(obj1.add.bind(obj2, "h", "y")()); // 28hy;

-

**Otipy interview**: 1st round (30-04-2022)

1. You know about Bootstrap?- Yes
2. Which library of the page layouting was being used?-Bootstrap
3. How responsive middle column to be made inside a 3-column table. Other two are non-responsive?
4. Difference between visibility & display?

Ans: display:none **means** that the tag in question will not appear on the page at all (although you can still interact with it through the dom). ... **visibility**:**hidden means** that unlike display:none , the tag is not **visible**, but space is allocated for it on the page. The tag is rendered, it just isn't seen on the page.

1. Difference Absolute and fixed?

Absolute is relative to the parent position on which relative position is given else highest parent <html> tag. A fixed position element is positioned relative to the *viewport*, or the browser window itself. The viewport doesn’t change when the window is scrolled, so a fixed positioned element will stay right where it is when the page is scrolled.This might be used for something like a navigation bar that you want to remain visible at all times regardless of the pages scroll position.

1. Html tags? What does aside do?

Ans: The <aside> [HTML](https://developer.mozilla.org/en-US/docs/Web/HTML) element represents a portion of a document whose content is only indirectly related to the document's main content. Asides are frequently presented as sidebars or call-out boxes.

1. Why section tag is used?

a semantic element for creating standalone sections in a web page. These sections should be made up of related content, like contact information

1. How to optimize the page loading? There are multiple methods? Tell me one or two?
2. Img tag & background img tag? Which one is better?

**<img>** tag:

1. It is an HTML element that explicitly embeds an image into the page.
2. Syntax: **<img src="path/to/image.jpg" alt="Image description" />**
3. Supports **alt** attribute, which provides a text description of the image for accessibility purposes.
4. The image is part of the document flow and can be styled using CSS properties like **width**, **height**, **border**, and more.
5. Can be used with responsive design techniques like **srcset** and **sizes** attributes to deliver different image resolutions based on the screen size.
6. Suitable for content images that have semantic meaning or convey information, such as photos, diagrams, or illustrations.

**background-image** CSS property:

1. It sets an image as the background of an HTML element.
2. Syntax: **element { background-image: url("path/to/image.jpg"); }**
3. Does not support the **alt** attribute, so it's not suitable for conveying important content or information.
4. The image is not part of the document flow and doesn't affect the layout or sizing of the element.
5. Provides more control over image positioning, scaling, and repetition with additional CSS properties like **background-position**, **background-size**, and **background-repeat**.
6. Suitable for decorative images, patterns, or visual effects that don't have semantic meaning or convey information.

Also, background img don’t get printed but img get printed.

1. Redux vs other state management? -ok

**CredAvenue: 1st Round (30-04-2022)**

1. Difference between Hoisting & scoping? – ok- check youtube
2. Definition of Higher order functions?

A “higher-order function” is a function that accepts functions as parameters and/or returns a function.

First order function: functions act as variable

1. Event bubbling?

Event bubbling is a method of event propagation in the HTML DOM API when an event is in an element inside another element, and both elements have registered a handle to that event. It is a process that starts with the element that triggered the event and then bubbles up to the containing elements in the hierarchy. In event bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements.

Opposite is Event capturing.

Default is event bubbling. Event capturing can be done by using below:

el.addEventListener("click", function () {

console.log("i run in capture");

}, true); //set true

1. Anchor tag with href and onclick function. how to prevent default?

Event.preventdefault()- check the code

1. Random number generator function

const ranNum = (a, b) => {

//a is included but b is not

// const min = a;

// const max = b;

//a & b both included

// const min = a;

// const max = b + 1;

// a & b both not included

// const min = a + 1;

// const max = b;

const randomNum = min + Math.trunc(Math.random() \* (max - min));

console.log(randomNum);

};

ranNum(3, 5);

1. Object flattening (using function):

const flatOb = (obj) =>{

for(let i=0; i<Object.keys(obj).length; ){

if(typeof obj[Object.keys(obj)[i]] === 'object' && !Array.isArray(obj[Object.keys(obj)[i]])){

const ks = Object.keys(obj[Object.keys(obj)[i]]);

const childOb = obj[Object.keys(obj)[i]];

for(let j=0; j<ks.length; j++){

obj[Object.keys(obj)[i]+'\_'+ ks[j]] = childOb[ks[j]];

}

delete obj[Object.keys(obj)[i]]

} else{

i++;

}

}

}

1. Html & Html5 difference?
2. What is semantic html and benefits?

-Search engine optimization, screen reader software benefits

1. Psudeo css & add comma using css

A CSS pseudo-element is used to style specified parts of an element.

Example: selector::pseudo-element {  
  property: value;  
}

::before, ::after , ::first-letter etc

Psuedo classes: are

Selector: hover, :active, :visited, : valid etc

1. Different types of lists (Ordered list, unordered list, descriptive list).

Ordered lists, which have an inherent order and each item is numbered.

Unordered lists, which have no inherent order and each item is bulleted.

Description lists, which contain a list of terms and descriptions for each term.

Example:

<dl>

        <dt>Python:</dt>

        <dd>It is a programming language</dd>

        <dt>C++:</dt>

        <dd>It is also a programming language</dd>

    </dl>

1. How bullet points are removed from list

List-style: none

text-decoration: none (link- to remove underline)

1. Border-box and \* mark usage in css?- ok (  box-sizing: border-box;)
2. @media query? - ok
3. Specificity in css?- ok

The following is the order of specificity, from least specific to most specific:

* Type selectors (e.g. **h1**, **p**, **div**)
* Class selectors (e.g. **.class**)
* ID selectors (e.g. **#id**)
* Inline styles (e.g. **style="color:red;"**)

**Otipy interview**: 2nd round (04-05-2022)

1. Can you convert class-based component to function based component? & vice versa?
2. Design a universal Button.
3. Design a universal popup.
4. How to send data from child to parent?
5. Loosely built components?
6. Explain the system design.
7. How many pages were there in your website?

**Quiz:**

1. Error boundaries can catch errors in a. event handlers b. null point exception c. server side rendering

Ans: Null point exceptions: basically when we are trying to read the property of null or undefined then this error appears.

1. What is time complexity of searching a key in dictionary/object? – O(1)
2. Functional component efficiency vs class based components efficiency?

In general similar but functional component can be considered little highly efficient. In class component there is additional memory consumption compared to functional as they have to keep the memory of instance properties and methods. In component method, there are multiple lifecycle methods.

Also, there is concurrent mode in modern react which makes functional components more efficient.

**Interview -5 (LeadSquared):**

Q. (function() {

    console.log(1);

    setTimeout(function(){console.log(2)}, 1000);

    Promise.resolve().then(() => console.log(5))

    setTimeout(function(){console.log(3)}, 0);

    console.log(4);

})();

Output: //

1 4 5 3 2

console.log(1);

console.log(2);

setTimeout(() *=>* {

  console.log(3);

});

new *Promise*((resolve) *=>* {

  console.log(4);

  setTimeout(() *=>* {

    console.log(5);

    resolve(true);

  });

});

Ans: 1 2 4 3 5

Q. sayOtherName() //

sayName()   //

var sayName =() =>{

    console.log('hello world')

}

function sayOtherName(){

    console.log('world is beautiful')

}

Ans: World is beautiful

Its not a function

**10-May Piramal:**

1. Make to do list with button and checkbox

**10-May Invenics:**

1. What is difference between virtual DOM & Real DOM?- ok
2. Why is react needed over Javascript? – Performance and smooth experience
3. Is single page application possible in javascript? - Yes
4. What are the React Hooks. Name some?- ok
5. How do you manage API calls? API call under API call through React?

MakeMytrip 1st round:

1. Why Promise is required? If promise is not used what would happen?

**Recro Interview:**

1. Revise Redux concepts- ok
2. React component input with object
3. Flatten the array

Method-1

for (*let* i = 0; i < arr1.length; ) {

*Array*.isArray(arr1[i]) ? arr1.splice(i, 1, ...arr1[i]) : i++;

}

Method-2

*const* flatten = (*arr*) *=>* {

*let* status, blank;

*let* newArr = *arr*.slice();

  do {

    blank = [];

    status = false;

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

      if (*Array*.isArray(newArr[i])) {

        status = true;

        blank = blank.concat(newArr[i]);

      } else {

        blank.push(newArr[i]);

      }

    }

    newArr = blank.slice();

  } while (status);

  return blank;

};

1. CSS optimization
2. Make CSS to below
3. <!DOCTYPE html>
4. <html>
5. <head>
6. <meta charset="utf-8">
7. <title>Sample Title</title>
8. </head>
9. <body>
11. <div class="container">
12. <div class="pill">
13. <span>L</span>
14. <span>Pill Label</span>
15. <span>X</span>
16. </div>
17. </div>
19. </body>
20. </html>
21. <!--
22. Write HTML for the pill
23. 1. Logo
24. 2. Label
25. 3. Cross Icon
26. 4. Should be responsive
27. Ref: https://www.lightningdesignsystem.com/components/pills/
28. -->

Check classes & object syntaxes

**Interview with “ofbusiness”:**

1. How Inheritance work? -ok
2. How async & defer work in javascript script loading?-

Page loading means: html parsing & script loading. (script loading means fetching of script from network & then executing line by line).

Without any attribute: pauses html parsing as soon as script is reached. Then it fetch script & then execute then resumes the html parsing

Async: script fecthing is done parallel to html parsing & then html parsing is paused and script is executed and then html parsing resumes. (async doesn’t guarantee the order of different scripts)

Defer: script are fetched in parallel to html parsing & executed after complete html parsing.

1. What is this keyword and how it is used? - ok
2. What is high order component and give an example?

Functional components: Functional components are functions that takes in props and return JSX.

Pure components: In React, a pure component is a type of component that only depends on its input props and doesn't have any side effects such as modifying state or interacting with the browser's DOM. A pure component is a function component that is optimized for performance because it avoids unnecessary re-renders. components which don’t re-render unless there is change in the props. (Same is achieved through React.memo in functional components). It compares the props as shoallow comparison.

Higherorder component: It takes a component and returns a new component.

Export A component like HOC(A)

Export A component like HOC(B)

Function HOC =(wrappedComponent)=>{

Function HOC {

useState

function increament{}

Return <wrappedComponent increament ={increament}/>

}

Return HOC

}

Stateless component, (presentation component, dumb component): it doesn’t have any internal state.

Stateful component(smart component): It has its own state component.

**Uncontrolled Components:**Uncontrolled Components are the components that are not controlled by the React state and are handled by the [DOM](https://www.geeksforgeeks.org/dom-document-object-model/) (Document Object Model). So in order to access any value that has been entered we take the help of refs.

**Controlled Components:**In React, Controlled Components are those in which form’s data is handled by the component’s state. It takes its current value through props and makes changes through callbacks like onClick, onChange, etc. A parent component manages its own state and passes the new values as props to the controlled component.

1. How redux is helpful?

Redux is useful for state management for complex statemanagement.

1. Why keys used in list element of react? What will happen in DOM if you don’t use list?

To help react identify the list items of array.

Otherwise, react will first compare the array length and render the all items of the array. Then update content of each item as per the method mentioned (e.g., ascending /descending etc). It will lead to performance issue and if some state is attached to particular list item it will get overwritten by other content leading to bug.

1. Why react is used? Why not angular or other thing? What made you choose react?
2. Benefit of strict mode?

will throw errors if you try to assign a value to an undeclared variable

We have to define the variable using let, const or var. Without strict mode, variables get defined and don’t give error. Also, strict mode does not allow us to use variables reserved (e.g. this).

Without strict mode, JavaScript fails silently without letting you know there's been an error. With strict mode enabled, JavaScript will throw errors, which can be helpful for debugging and learning from your mistakes.

Check if strict mode is used in cornerstone

**Interview with PureSoftware:**

1. Difference between shallow copying & deep copying

Shallow copy of objects methods:

Object.assign(target, source)

{…object}

For deep copy of objects:

* 1. Const object2= JSON.parse(JSON.stringify(object)) //problem is that 1. if object is having function then it will be completely missed in the new object 2. If date is used in object1 then in object 2 its type will become string
  2. Install loadash library and use the cloneDeep method
  3. Make a for loop:

const arrClone = arr =>{

const newArr =[];

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

if(Array.isArray(arr[i])){

newArr[i] = arrClone(arr[i]);

} else if(typeof arr[i] === 'object'){

newArr[i] = objClone(arr[i]);

} else{

newArr[i] = arr[i];

}

}

return newArr;

}

const objClone = (obj) =>{

const newObj ={};

const ks = Object.keys(obj)

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

if(Array.isArray(obj[ks[i]])){

newObj[ks[i]] = arrClone(obj[ks[i]]);

} else if(typeof obj[ks[i]] === 'object'){

newObj[ks[i]] = objClone(obj[ks[i]]);

} else{

newObj[ks[i]] = obj[ks[i]];

}

}

return newObj;

}

Shallow copy is used to copy the main object but nested object are not copied, they are still reference

**Copy the function:**

const fn = (a,v) =>{

return a+v;

};

Function.prototype.clone = function(){

let that = this;

const temp = function (){

return that.apply(this,arguments)

}

return temp

}

const newFn = fn.clone();

1. What is Polyfilling? - ok
2. Perform sum(1)(2)(3)(4) - ok
3. What is prototypes? - ok
4. How to select an element next to an element using css?

div + p : Select and style the first <p> element that are placed immediately after <div> elements:

div >p : It will target elements which are DIRECT children of a particular element.

div ~ p: it will select all the elements after div within same parent.

div p : all p child element inside div parent. And even grandchildren as p tag will be selected

div,p: applicable on each

.box .fox(with space): child class fox within child class box

.box.fox(without space): class=”box fox”

1. What is the purpose of :after in css?

Ans: In CSS, ::after creates a [pseudo-element](https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements) that is the last child of the selected element. It is often used to add cosmetic content to an element with the [content](https://developer.mozilla.org/en-US/docs/Web/CSS/content) property. It is inline by default.

/\* Add an arrow after links \*/

a::after {

content: "→";

}

1. Show a usecase of redux. - done
2. What are the error boundaries in class based components.

They provide a way to gracefully handle errors in the UI, instead of crashing the entire application or displaying a broken interface to the user.

1. **static getDerivedStateFromError(error)**: This lifecycle method is called when an error is thrown in a descendant component. It receives the error as a parameter and should return an object to update the state. This method is used to render a fallback UI after an error has occurred.
2. **componentDidCatch(error, info)**: This lifecycle method is called after an error has been thrown in a descendant component. It receives the error and additional information as parameters. This method is typically used to log the error or report it to an error tracking service.
3. To use the error boundary, wrap it around any components that may throw errors:

<ErrorBoundary>

<ComponentThatMayThrowError />

</ErrorBoundary>

When an error occurs in **ComponentThatMayThrowError**, the error boundary will catch the error, update its state, and render the fallback UI. Other parts of the application that are not wrapped by the error boundary will continue to work normally.

Keep in mind that error boundaries only catch errors in the components below them in the component tree. They do not catch errors in their own rendering process, lifecycle methods, or constructors. To catch errors in the error boundary component itself, you would need to wrap it with another error boundary.

1. Difference between useMemo() & useCallback(). Show a usecase of useCallback. -ok

**Interview with squa.re**

1. What were the modules used in your website?
2. React is server side rendering? What is server side rendering? What is SSR?
3. What is Code splitting?
4. Did you used next.js? - No
5. What is linear complexity?
6. What is middleware in redux? Why these are used? Thunc, Saga etc?

done

1. What is context API? Why it is used? When redux is used?

Context API is used to avoid prop drilling and make nested components availabily of access of parent state, & its management function. It is built-in API of react & should be used for very less changes such as logged in status, color theme etc.

Redux is purely state management tool & it’s a external library. used for high frequency changes in states.

1. What is props drilling?- ok
2. Explain about the functions take Every & ? Used in reactsaga
3. Explain about React.memo, useCallback & useMemo

Ans: React.memo is used to avoid unnecessary re-rendering by comparing the props change. If change is there in props then only component is re-rendered. But it comes at cost of props comparison process.

Export default React.memo(Component);

useCallback(): If functions are passed as props, then Functions are always re-created in each re-rendering of the parent component & react.memo treats it as new function as memory reference is different.

So, useCallback() memorize the function and re-creates only if dependency is changed

Const memorizedFn = useCallback(( ) => {

Original function;

} ,[array of dependency])

useMemo( ): if Object or array are passed as props and there is some performance intensive work in child component using that array, but as array are not primitive value; so each re-rendering of parent component will prompt the re-rendering of the child component & re-evaluation of that performance intensive task. So, to avoid that,

useMemo is used to memorize the array in parent :

const items= useMemo(( ) =>{

return array // whichever needs to be returend

},[array of dependency]);

In child component, output of that performance intensive task can be memorized:

Const sortedItems = useMemo(( ) =>{

Return items.sort((a,b)=> return a-b);

},[items])

All above comes at the cost of memory consumption to memorize the functions,array, object. Also comparison of the props. So, these should be used only if there is intensive performance work in child comp.

**Interview with Concentrix:**

1. Whar is web worker in HTML? – In HTML, a web worker is a JavaScript script that runs in the background, independently of the main thread, allowing for the creation of multi-threaded web applications. This is important because JavaScript, which is the primary language used in web development, is single-threaded by default, meaning that only one task can be executed at a time. By using web workers, developers can run time-consuming and CPU-intensive tasks, such as parsing large amounts of data, without blocking the main thread and affecting the responsiveness of the user interface. Web workers also enable the sharing of data between multiple threads, allowing for efficient communication and coordination between them.

To create a web worker in HTML, you need to create a separate JavaScript file that contains the code for the worker, and then use the Worker() constructor to instantiate the worker and start it running. The main thread can then communicate with the worker using the postMessage() and onmessage methods.

In your React component, import the Worker object from the worker-loader package, which allows you to load the worker file as a module:

1. What is pure component in react?- ok
2. How many input types are there? - ok
3. How will you find the geoposition of user using html?- **navigator.geolocation.**getCurrentPosition(callbackFn(position) => {console.log(position.coords))
4. How will you find location of user in webpage using html? – window.getSelection()

**Interview with SirionLabs**

1. var & let differences?- ok
2. const a= ['hello']

a.push('hi') //will this work? Ans yes as a is still pointing to same reference in memory.

1. Show example of closure?
2. Output of below:

(function immediateA(a) {

return (function immediateB(b) {

console.log(a); // What is logged? Ans: 0

})(1);

})(0);

1. for (var i = 0; i < 3; i++) {

setTimeout(function log() {

console.log(i); // What is logged? Ans: 3 3 3

}, 1000);

}

1. How to solve without using let. Use closure:

for (var i = 0; i < 3; i++) {

(function () {

var j = i;

setTimeout(function log() {

console.log(j); // What is logged? Ans: 0,1,2

}, 100);

})(i);

}

1. What is difference between call, bind & apply?- ok
2. const object = {

message: "Hello, World!",

logMessage() {

console.log(this.message);

},

};

object.logMessage(); // What is logged? Ans: ‘Hello World’

setTimeout(object.logMessage, 100); //What is logged? Ans: undefined

const logMessageOutside = object.logMessage;

logMessageOutside() ////What is logged? Ans: undefined

Why undefined?

Ans: Detaching and calling methods like this means they no longer apply to the objects they were created on.

setTimeout(object.logMessage.bind(object), 100);

logMessageOutside.apply(object)

**Delhivery 17-06:**

1. What is difference between session storage, localstorgae & cookies & caches?
2. What is DOM ? : Document Object Model
3. Function this keyword
4. Promise ?

**TicketMaster:**

1. What is use of webpack & babel?

Babel transpile & polyfills the code back to ES5

1. What is slice in redux toolkit?- done.
2. Method to optimize react app?

What is Browserify?- brings the power of modules to be used in browsers e.g. crypto: require.resolve('crypto-browserify'),

timers: require.resolve('timers-browserify'),

stream: require.resolve('stream-browserify'),

Browserify is a JavaScript tool that allows you to use Node.js-style require statements to organize your code into modules and bundle them for use in the browser. Essentially, it brings the power of Node.js module system to client-side JavaScript, enabling developers to write modular code that can be easily managed and reused.

1. How did you test the react app? What were the test cases?
2. Different types of props in react?
3. What is re-conciliation in react?

Ans. Reconciliation is the process through which React updates the Browser DOM.

Important concepts behind the working of the Reconciliation process are:

Virtual DOM  & Diffing Algorithm

**Paytm Round-1:**

* 1. Webpack versions? And related things.
  2. Lighthouse? (Optimization)
  3. Server side rendering
  4. React V18 -explain later
  5. Why doctype is defined?

The **<!DOCTYPE>** declaration is used in HTML to inform the browser about the version of HTML being used in the document. It should be placed at the very beginning of an HTML document, before the opening **<html>** tag. The **<!DOCTYPE>** declaration is not an HTML tag, but rather an instruction to the web browser. f you do not include the **<!DOCTYPE>** declaration in your HTML document, the browser might render the page in what is known as "Quirks Mode." When a browser is in Quirks Mode, it may use inconsistent or outdated rendering rules, which can lead to unpredictable behavior, visual inconsistencies, and layout issues. By including the **<!DOCTYPE html>** declaration in your HTML document, you tell the browser to use "Standards Mode," which provides a more consistent, modern, and standards-compliant rendering environment.

* 1. What is meta tag in html?

Meta tag is for metadata for which tags are not available, like name, organization etc. Meta tags are important because they provide information about the document that can be used by browsers, search engines, and other web services to better understand the content, display it correctly, and improve the user experience. For example character encoding (UTF-8), viewport setting,

* 1. Difference between sessionstorage, localstorage, cookies and cached
  2. What happens if a tab is opened and data is stored in sessionstorage. And without closing the tab, new website is opened there. But then again last page is opened. Will sessionstorage be available or removed? -will stay

**Interview with Credgenics:**

1. What is polyfilling? -ok
2. What is prototyping and make your own map function.

Array.prototype.myMap = function(callbackFn){

const ans = [];

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

ans.push(callbackFn(this[i],i,this))

}

return ans;

}

const arr = [1,2,3,4]

const printArr = arr.myMap((el,i,arr)=>{

return el+1;

})

console.log(printArr)

1. var a ={'key': 'a'}, b= {'key': 'b'}, c={'key': 'c'}

a[b] = 123

a[c] = 456

console.log(a[b]) //456

Because of Object to primitive value conversion.

The object-to-primitive conversion is called automatically by many built-in functions and operators that expect a primitive as a value.

There are 3 types (hints) of it:

* "string" (for alert and other operations that need a string)
* "number" (for maths)
* "default" (few operators, usually objects implement it the same way as "number")

The specification describes explicitly which operator uses which hint.

The conversion algorithm is:

1. Call obj[Symbol.toPrimitive](hint) if the method exists,
2. Otherwise if hint is "string"
   * try calling obj.toString() or obj.valueOf(), whatever exists.
3. Otherwise if hint is "number" or "default"
   * try calling obj.valueOf() or obj.toString(), whatever exists.

All these methods must return a primitive to work (if defined).

In practice, it’s often enough to implement only obj.toString() as a “catch-all” method for string conversions that should return a “human-readable” representation of an object, for logging or debugging purposes.

1. What is event delegation?

**Interview with Nagarro (After test round):**

1. Semantic HTML & its usage? -ok
2. How many different ways you can center the child w.r.t. Parent? - done above.
3. What is generator? Have you worked on react saga? – check study notes word file
4. Unit testing? Read about it. -ok
5. Different ways of react optimization? Optimized way to load images?

React optimization involves making your React application run faster and more efficiently. There are several ways to optimize your React application, and an optimized way to load images. Here's a list of different optimization techniques and the best practice for loading images:

1. Code Splitting: Break down your application into smaller chunks using dynamic imports and React.lazy(). This reduces the initial load time by only loading the necessary components.
2. Memoization: Use React.memo() for functional components and PureComponent for class components to avoid unnecessary re-renders. This technique stores the results of expensive function calls and returns the cached result when the same inputs are provided.
3. Debouncing and Throttling: Limit the frequency of function calls, especially for event handlers such as scroll or resize events, by implementing debounce or throttle functions.
4. Using keys: When rendering a list of elements, always use unique keys to help React identify which items have changed, are added, or are removed. This can reduce the number of DOM operations and improve performance.
5. Profiling and analyzing: Utilize React DevTools to identify performance bottlenecks in your application and find opportunities for optimization.

Optimized way to load images:

1. Lazy Loading: Load images only when they are in or near the viewport using Intersection Observer API or a third-party library like react-lazyload.
2. Image Optimization: Compress images and use modern file formats like WebP or AVIF to reduce file size without compromising on quality.
3. Responsive Images: Use the **srcset** attribute and **picture** element to serve the right-sized image based on the user's device and screen resolution.
4. Preloading: For critical images or assets, use the **preload** attribute in the **link** tag to fetch them ahead of time, improving the overall user experience.
5. Using a Content Delivery Network (CDN): Host images on a CDN to reduce latency and improve load times for users located in different geographical locations.
6. Caching: Utilize proper caching headers and strategies to cache images on the user's browser, reducing the need for repetitive downloads.

By combining React optimization techniques with image loading best practices, you can significantly improve the performance and user experience of your application.

1. Pre-processor of SASS? How do you use it? How does it work? -ok
2. Webpack usage? – check study notes
3. Parcel usage?- webpack is more flexible and configurable but required steep learning curve. Parcel and Webpack are both web application bundlers that serve similar purposes but have different approaches. Parcel focuses on simplicity and zero-configuration, while Webpack offers more flexibility and control. Developers can choose between these tools based on their project requirements and personal preferences.
4. How build is made? - check study notes of webpack
5. How deployment is done?- In summary, deploying a website involves building your site, choosing a hosting provider, configuring your domain name, uploading your website files, and testing your site.
6. What is pure function? And where is it used in redux?

Pure function: Any function which doesn’t have side effect. Basically for a certain input there will be same output. It is used for reducer functions.

1. Usage of typescript? -done
2. There is an array. How many ways you can make the array an empty array?
   1. Change the arr.length = 0;
   2. Use arr.pop() or arr.shift()
   3. Use arr.splice()
3. Work on advanced Javascript & react.
4. NavLink vs Link
5. D3.js & NVD3 -ok
6. Promises- ok
7. Get vs put vs post¨ In summary, GET is used to request data from a server, PUT is used to update existing data on the server, and POST is used to submit new data to be processed and stored by the server.
8. JSON Web Token:
9. React router.
10. React V18 features:

basically focused on concurrent rendering (basically pause some work and work on more important task and continue.

ReactDom.create()

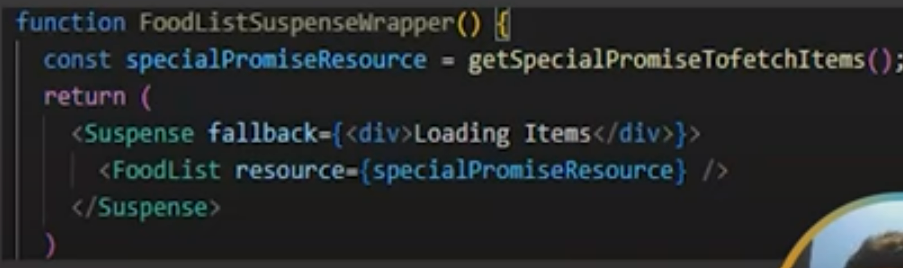
Automatic batching: all state update rendering will be called in batching (similar to current automatic batching on event handler)

Transition: startTransitions(()=>{

//state update which is not so much having priority.

}) – if some urgent task comes then it will be pause this and finish rugent task. Then resume this task

Suspense on the server:



NewHooks:

useId: id =useId() – it creates a random id at each render which can be used as id of input tags etc(to be used for id only, not for key)

UseTransition:

Const[isPending, startTransition] = useTransition(); //isPending is true if data is loading

UseDeferredValue

useSyncExternalStore

useInsertion

useStrict:

1. Next.js
2. Check table code in csod

**Nagarro Final round:**

1. useCallback & useMemo? And their usage?
2. How will you optimize the react app?-
3. Code splitting: One way to optimize a React app is to implement code splitting. This involves breaking your application into smaller, more manageable chunks that can be loaded as needed. Have you used code splitting before or are you familiar with tools like React.Lazy and React.Suspense?
4. Minify and compress: Minifying and compressing your code can significantly reduce the file size of your application, leading to faster load times. Have you used any tools to minify or compress your code, like Terser or Webpack?
5. Optimize images and assets: Large images and other assets can slow down your application. Do you know how to optimize images and other assets, such as using compression tools, responsive images, or serving them from a Content Delivery Network (CDN)?
6. Remove unused code: Unused code, also known as dead code, can bloat your application unnecessarily. Are you familiar with tools like Webpack's Tree Shaking feature or PurgeCSS for removing unused CSS?
7. Use Pure Component or React.memo: These can help prevent unnecessary re-renders of components when the props or state have not changed. Have you used PureComponent or React.memo in your projects?
8. Use the production build: The development build of React includes extra warnings and checks, which can slow down your application. Are you aware of how to create a production build of your React app?
9. Performance profiling: Profiling your application's performance can help identify bottlenecks and areas that need optimization. Are you familiar with the React Developer Tools and its Profiler tab?
10. Optimize API calls: Reducing the number and frequency of API calls can help improve your application's performance. Do you know how to use techniques like throttling, debouncing, or caching to optimize API calls?
11. Compression: Compressing image files reduces their file size, which can speed up the loading of your application. There are various tools and libraries available for compressing images, such as ImageOptim, TinyPNG, and jpegoptim. Ensure you're using the right format for your images, such as WebP, which provides better compression than JPEG or PNG.
12. Responsive images: Using responsive images means serving different image sizes based on the user's device and screen resolution. This can help reduce the amount of data that needs to be loaded, improving performance. You can use the **srcset** attribute or the **picture** element in HTML, or the **react-responsive-image** library in React.
13. Lazy loading: Lazy loading defers the loading of off-screen images until they are needed, reducing the initial page load time. You can implement lazy loading using the **loading="lazy"** attribute in the **img** tag (for modern browsers) or with libraries like **react-lazyload** and **react-progressive-image** for broader compatibility.
14. Inline small images: You can inline small images directly in your CSS or JavaScript as Base64-encoded data URIs. This can help reduce the number of HTTP requests and speed up the initial rendering of your application. However, this technique should be used sparingly, as it can increase the size of your CSS/JS files.
15. Use SVGs for icons and simple graphics: Using SVG (Scalable Vector Graphics) for icons and simple graphics can lead to smaller file sizes and better performance, especially on high-resolution screens. SVGs are also easier to scale and style with CSS.
16. Content Delivery Network (CDN): A CDN is a network of servers distributed across different locations that can serve your application's assets, like images, stylesheets, or scripts, to users from the server closest to them. This can significantly improve loading times for users located far away from your primary server. Popular CDNs include Cloudflare, AWS CloudFront, and Akamai.
17. How private routes are set based on authorization (authentication already done)? Any lib used?
18. What is error boundary? -ok
19. Why redux is used? Why we cannot manage store as global variable on window object?- single source of truth

**LinkSoft:**

1. Debouncing, currying, array flattening etc.
2. iframe in html?
3. Responsive vs fluid css?  
   Fluid CSS: also known as liquid layout, involves designing layouts where elements resize dynamically based on the size of the viewport or container. Instead of using fixed units like pixels for sizing elements, fluid layouts use relative units such as percentages or viewport units (vw, vh) to specify dimensions  
   **Responsive CSS:** Responsive CSS focuses on designing layouts that respond to changes in screen size by adjusting the layout and content presentation. This typically involves using media queries to apply different styles based on the width of the viewport.
4. P tag & span tag difference?
5. Canvas element in html  
   The HTML <canvas> element is used to draw graphics on a web page via JavaScript. It provides a drawable region defined in HTML code with <canvas> and exposes a rendering context that can be used to draw graphics. Here’s a basic overview of how to use the <canvas> element:

<canvas id="myCanvas" width="500" height="500"></canvas>

const canvas = document.getElementById('myCanvas');

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

// Drawing a rectangle

ctx.fillStyle = 'blue';

ctx.fillRect(10, 10, 100, 100);

// Drawing a circle

ctx.beginPath();

ctx.arc(150, 75, 50, 0, 2 \* Math.PI);

ctx.stroke();

1. Accessibility in react? (Basically screen reader)
2. What is aria-label?
3. Why strict mode is to be used in react?

**CleverTap:**

1. Make pagination component

**MoneyView:**

1. Const str = ‘saurabh’

Str.toString()

How are we able to access the toString() method?  
  
In JavaScript, even though **str** is declared as a string literal, when you try to access a property or method on it, JavaScript automatically converts it into a String object. This process is called "implicit coercion" or "auto-boxing".

So, when you call **str.toString()**, behind the scenes, JavaScript converts the primitive string **str** into a String object, allowing you to access the **toString()** method which is inherited from the **String** object's prototype.

This behavior is part of JavaScript's dynamic nature and is one of the features that make the language flexible and easy to use.

// function Shape() {

//   this.x = 0;

//   this.y = 0;

// }

// Shape.prototype.move = function (x, y) {

//   this.x += x;

//   this.y += y;

// };

// function Rectangle() {}

// Rectangle.prototype = Object.create(Shape.prototype);

// Rectangle.prototype.constructor = Rectangle;

// var rect = new Rectangle();

// rect.move(1, 1);

// console.log("x:", rect.x);

**Naehas-31-Jan-2024 (FloCareer)**

1. Difference between passbyValue & passByReference?
2. What all are other non-primitive variables in JS apart from object and array?

Function, Maps, Sets, Date, RegExp, Symbol, BigInt, Promise Object, Error Object

1. Difference between Promise and async?
2. Difference between Promise.allSetteled and Promise.all?
3. Difference between class based and functional based react?
4. What all things you have used in react?
5. What precautions/guidelines we use while making a custom hook?
6. Write counter in react component
7. // swap the strings without using another variable

let a ='str';

let b = 'random';

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

if(i<a.length){

b = b+ a[i];

}else if(i === a.length){

a=b[0];

b= b.slice(1);

} else{

a=a+b[0];

b= b.slice(1);

}

}

console.log(a,b);

**HDFC:**

1. What is hoisting?
2. Difference between angular and react?
3. Why do we use the error-boundaries?
4. What is server side rendering?
5. What is call, bind apply method?
6. What are mutable and immutable objects? What is Object.freeze?
7. Why Object.entries, Object.keys and Object.values methods exist? Interchange the values using code.
8. What is progressive web app (pwa) and single page design?
9. Any project made from scratch? Architectural decision did you make?
10. Write your own call, bind, apply method?
11. What is props drilling?
12. What are the loopholes in context api?
13. Write a code snippet to pass on the value from child to parent.

**HDFC other round:**

console.log("begins"); //

setTimeout(() => {

console.log("setTimeout 1");

Promise.resolve().then(() => {

console.log("promise 1");

});

}, 0);

new Promise(function (resolve, reject) {

console.log("promise 2");

setTimeout(function () {

console.log("setTimeout 2");

resolve("resolve 1");

}, 0);

}).then((res) => {

console.log("dot then 1");

setTimeout(() => {

console.log(res);

}, 0);

});

//Ans: begins -> promise2 -> setTimeout1 -> promise 1 -> setTimeout 2 -> dot then 1 -> resolve 1

**Impetus(22-Jun)**

1. How objects can be created ? (multiple ways?) difference between const = new Object () & const obj = {}
2. Make a custom hook used for apis?
3. Output of below?  
   console.log([1,2,3,4,5,6].map((m, i) => { if (m%2 === 0) return m; } )); //[]  
   console.log([1,2,3,4,5,6].filter((m, i) => { if (m%2 === 0) return m; } )) // [2,4,6]

Difference between map, filter and reduce? Why different methods are made? (not just functionality)-   
Probable Ans:map is used for transformation. It should be used only when value is returned.

1. Write context api syntax.
2. <span><div></div></span> => why we don’t use? Ans: (inline and block elemets)
3. Header and footer element?
4. HTML apis? (navigator.geolocation, Dom api (e.g. getElementbyID), WebStorageApi(e.g. localstorage.getItem), fetch api, file reader, websocket) etc
5. Web storage and web worker and service worker?
6. React testing library – (RTL exp)?
7. How response layout is achieved using bootstrap?
8. Performance enhancement using webpack? Sonar?
9. Singleton design , MVC design.:

The Singleton pattern is a design pattern that ensures a class has only one instance and provides a global point of access to it. This is useful when exactly one object is needed to coordinate actions across a system.

class Logger {

constructor() {

if (Logger.instance) {

return Logger.instance;

}

this.logs = [];

Logger.instance = this;

}

log(message) {

this.logs.push(message);

console.log(`LOG: ${message}`);

}

printLogCount() {

console.log(`${this.logs.length} Logs`);

}

}

const logger = new Logger();

Object.freeze(logger); // Ensure no modifications to the instance

export default logger;

model, view controller (MVC)design. Not so significant for react. But example:   
e.g.View: UI component (rendered with control of props)  
model: api actions, redux management etc   
controller: main component dispatching actions, and renderin UI

1. Jest cases

**Shivam Interview qn:**

1. React framework vs library?
2. React portal
3. Meta tags
4. http vs https
5. Where redux stores the data?
6. How to call an api without using fetch and axios? Ans: XMLHTTPRequest
7. useReducer hooks
8. git merge and git rebase
9. package.json and package-lock.json. What happens if we add package-lock.json to gitignore?
10. Shadow DOM vs Virtual DOM?
11. IFFE and reconciliation?
12. How to cancel Promise?
13. JS functions are call by value or call by reference?
14. Hashrouter vs BroswserRouter?
15. What is temporal dead zone?
16. Why setState in react is async?
17. What are iframe?
18. Variable in css?
19. Dfferene between null, undefined and NaN?
20. +0 === -0  
    +0 == -0

**CourseFinder.ai**

1. Write a function that takes a string as input and returns the length of the longest substring that contains no repeated characters.

// Example 1: Input: s = "abcabcbb" Output: 3

// Example 2:Input: s = "bbbbb" Output: 1

// Example 3: Input: s = "pwwkew" Output: 3

function checkSub(s){

let max =0;

let tempS ='';

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

if(tempS.includes(s[i])){

max = Math.max(tempS.length,max);

const index = tempS.indexOf(s[i]);

tempS = tempS.substring(index+1) +s[i];

} else{

tempS += s[i];

}

}

console.log(Math.max(max, tempS.length));

return Math.max(max, tempS.length);

}

1. Write a task manager with edit and delete functionality.
2. Create a layout like an e-commerce website search results.Requirement:There should be 4 items per row on the desktop web and 2 items per row on mobile.
3. What is CMS?
4. How do we fetch data from backend in react?

**Others:**

1. how do you optimize the webpage for print?

In your CSS, use the **@media print** media query to define print-specific styles.

@media print {

/\* print-specific styles go here \*/

}

@media print {

.navbar,

.sidebar,

.footer {

display: none;

}

}

@media print {

body {

font-size: 12pt;

line-height: 1.5;

color: black;

}

} @media print {

h1, h2, h3 {

page-break-after: avoid;

}

. page-break {

page-break-before: always;

}

}

@media print {

img {

max-width: 100%;

page-break-inside: avoid;

}

. hide-on-print {

display: none;

}

. grayscale {

filter: grayscale (100%);

}

}

**Learning:**

1. For loop:

outer: for (let i = 0; i < 3; i++) {

for (let j = 0; j < 3; j++) {

let input = prompt(`Value at coords (${i},${j})`, '');

// if an empty string or canceled, then break out of both loops

if (!input) break outer; // (\*)

// do something with the value...

}

}

alert('Done!');

1. alert('1' + 2 + 2); // "122" and not "14"

alert(2 + 2 + '1' ); // "41" and not "221"

let user = {

name: "John",

age: 30,

isAdmin: true

};

for (let key in user) {

alert( key ); // name, age, isAdmin

alert( user[key] ); // John, 30, true

}

* [Object.getPrototypeOf(obj)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getPrototypeOf) – returns the [[Prototype]] of obj.
* [Object.setPrototypeOf(obj, proto)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/setPrototypeOf) – sets the [[Prototype]] of obj to proto.

The only usage of \_\_proto\_\_, that’s not frowned upon, is as a property when creating a new object: { \_\_proto\_\_: ... }.

Although, there’s a special method for this too:

[Object.create(proto, [descriptors])](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create) – creates an empty object with given proto as [[Prototype]] and optional property descriptors.