1. **What is ECMAScript? (European Computer Manufacturers Association Script)**

ECMAScript (ES) is a standard that defines the rules and features of JavaScript. It ensures that JavaScript works the same way across different platforms and browsers. Think of it as the blueprint for JavaScript, telling it what it can do and how it should behave.

1. **ES6 features**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Template String**

-> Template string is denoted by back tick operator `` and it is used for avoid string concatenation operator

If we string values using traditional and if we want to add some dynamic values in string at run time using variable name then we need to concat variable in string by “”+ so if we want to avoid this approach so we can use the template string concept shown in following code.

1. **Arrow Function**

-> An **Arrow Function** is a shorter and more concise way to write functions in JavaScript, introduced in **ES6 (ECMAScript 2015)**. It uses the => syntax (hence the name "arrow") and is especially useful for writing shorter callback functions.

1. **Rest operator**

-> REST Operator is used for pass infinite number of parameter to function called as REST Operator.

1. **Spread operator**

-> Spread operator is used for array destructing purpose

Array destructing means we can copy the array values in normal variable without giving the manually index values.

1. **OOP concept using JavaScript**

**6. Object literals**

-> Object literals indicate the we can declare java script and can store different types of data in it. In the form of key and value pair.

**Syntax:**

let variablename={

key:value

}

1. **Web services + REST API & JSON**

* **API** stands Application Programming Interface and API used by web services for provide

Communication between two applications those may be developed in same technology or may in different technology.

-> **JSON** stands for JavaScript Object notation but basically it is data exchange format and

It is language independent format means can work with any language and it is light

Weighted format as well as json store in the form of key and value pair and key of json object is always in the form of string and data may be any type.

1. **AJAX**

**AJAX is a technique for creating asynchronous web applications. It allows web pages to fetch data from a server without reloading the entire page. This improves user experience and makes applications more dynamic and responsive.**

-> AJAX stands for Asynchronous JavaScript and XML

Note: it is not language just it is technology

**Q. Why Ajax use?**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Update the web page without reloading it.

2. Perform partial updation on web page just update the some specified portion of web page.

3. Send data to server in background

1. **Async & Await**

**async Keyword:**

* The async keyword is used to declare an **asynchronous function**.
* An **asynchronous function** always returns a **Promise**. If the function has a return value, it is automatically wrapped in a Promise. If there is an exception thrown, the function will return a rejected Promise.

**await Keyword:**

* The await keyword can only be used inside an async function.
* It pauses the execution of the async function until the Promise is resolved or rejected. It makes the code wait for the result of the Promise and then moves to the next line.
* If the Promise resolves, the value is returned; if it rejects, an error is thrown.

1. **Promises**

-> In JavaScript, **Promises** are a modern and powerful way to handle asynchronous operations. They allow you to write cleaner, more readable code when dealing with asynchronous tasks such as API calls, file reading, or timeouts.

**What is a Promise?**

A **Promise** is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

It has three states:

1. **Pending**: The initial state, neither fulfilled nor rejected.
2. **Fulfilled**: The operation completed successfully, and the promise has a resulting value.
3. **Rejected**: The operation failed, and the promise has a reason for failure.
4. **Fetch function**

-> Fetch function is JavaScript function which is used for fetch API or fetch resource from a server and this function return object of promises

1. Regular Expression

-> A **Regular Expression** (commonly abbreviated as **regex** or **regexp**) is a sequence of characters that defines a search pattern. It is used to match, search, and manipulate text based on specific patterns.

1. How many data types in JS?

-> **Primitive Data Types**: String, Number, Boolean, undefined, null, Symbol

 **Reference Data Types**: Object, Array,function

1. **How to declare a variable in js?**

->1)var -function level scope, globally scope

**Reassignable**: You can reassign a variable declared with var.

2)**let**-block lavel scope

Not redeclaration but reassign

3)const-block level scope

Not reassign and red claration

1. **What is scope of variable?**

**Global Scope:**

* If a variable is declared outside of any function or block, it has global scope.

Ex. Var

**Function Scope:**

Ex. Var A variable declared inside a function is **function scoped**.

**Block Scope:**

* A variable declared inside a block

Variables declared with let and const inside a block are block scoped.

1. **Hoisting**

hoisted to the top, but the value is assigned later in the code

1. Equality operators in js

**=== (Strict Equality Operator)**

* Compares **two values** for equality **without converting their types** (no type coercion).
* Both the value and the type must be the same to return true.

**== (Equality Operator)**

* Compares **two values** for equality **after converting them to a common type** (type coercion).
* Performs type conversion, so it may produce unexpected results in certain cases.

1. Array methods

**1)push()**Adds one or more elements to the **end** of an array and returns the new length of the array.

2) **pop()** Removes the **last** element from an array and returns it.

3) **shift()** Removes the **first** element from an array and returns it.

4) **unshift()** Adds one or more elements to the **beginning** of an array and returns the new length of the array.

5) **map()** **transform an array** by applying a function to each element of the array and returning a **new array** with the results.

4) Filter() create new array and return only those elements they can passes a specific condition

Some guess the output questions

React

**1. What is React?**

-> React is an open JavaScript library that is used for build user interface (UI) for website and web application. Basically React is used for develop the single page application with multipage user interface

**2.Virtual DOM**

-> **Virtual DOM:** React Virtual DOM improved the performance by updating components that have changed rather than refreshing entire DOM this leads to smooth user experience

1. **What is state in react and why it is used?**

**What is State in React?**

In React, **state** is an object that stores data or information about a component. It determines how a component behaves and what it renders. State is **mutable** (it can be changed) and is managed within the component, allowing it to respond dynamically to user interactions, API calls, or other events.

**Why is State Used?**

State is used to make React components interactive and dynamic. It allows components to:

a) React to User Interactions

b) render dynamically

1. **Lifecycle of a component**

**Phases of a React Component Lifecycle**

React components go through three main phases:

1. **Mounting**: When the component is created and added to the DOM.
2. **Updating**: When the component is re-rendered due to changes in props or state.
3. **Unmounting**: When the component is removed from the DOM.

**Summary of Lifecycle Methods**

| **Phase** | **Method** | **Usage** |
| --- | --- | --- |
| **Mounting** | constructor() | Initialize state and bind methods. |
|  | getDerivedStateFromProps() | Sync state with props. Rarely used. |
|  | render() | Render the UI. |
|  | componentDidMount() | Perform side effects (e.g., API calls). |
| **Updating** | getDerivedStateFromProps() | Sync state with props. Rarely used. |
|  | shouldComponentUpdate() | Optimize performance by preventing re-renders. |
|  | render() | Render the updated UI. |
|  | getSnapshotBeforeUpdate() | Capture DOM info before updating. |
|  | componentDidUpdate() | Perform side effects after updates. |
| **Unmounting** | componentWillUnmount() | Cleanup before the component is destroyed. |

**Q. What is component?**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Component is a section or block of UI code in react and it is created by using class or function

**How many ways to create component in React or Types of component**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**There are two types of component in React**

**1. Class component**

**2. Function component**

**In React, state is a JavaScript object that stores information about a component.**

**5) Hooks**

**What is Hooks?**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Hooks are the some functions which help to manage the state variable as well as implement the all life cycle feature of class component using function component

Means we can say hooks help us to convert stateless function component to stateful function component.

**If we want to work with hooks in React we have some important points**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Hooks can only be called inside React function components

2. Hooks can only called at the top level of a component

3. Hooks cannot be conditional

**Router**

React Router is a JavaScript library that manages routing in React applications. It allows users to create single-page apps that can navigate without refreshing the page

**State**

A **state variable** is a variable that stores information about the current state of an application or component

**What is jsx**

JSX stands for JavaScript XML. JSX allows us to write HTML in React. JSX makes it easier to write and add HTML in React.

Types of Hooks

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1) useState**

* useState() hooks is used for manage the state variable using a function component Basically it is function which contain two parameters on is variable which is used for hold the value and second is function which is responsible for update the state variable

**Benefits of useState**

* Simplicity: Easy to use in functional components.
* Reactivity: Automatically re-renders components when the state changes.
* Flexibility: Manages primitive, array, or object data types.

Ex.

import React, { useState } from "react";

import "./styles.css";

export default function App() {

  const [count, setcount] = useState(0);

  let Increase = (e) => {

    setcount(count + 1);

  };

  let Decrease = (e) => {

    setcount(count - 1);

  };

  let Clare = (e) => {

    setcount(0);

  };

  return (

    <div className="App">

      <h1>Count:{count}</h1>

      <button onClick={Increase}>Increase</button>

      <button onClick={Decrease}>Decrease</button>

      <button onClick={Clare}>Clare</button>

    </div>

  );

}

**2. useEffect**

the useEffect() perform side effects in function component like as data fetching, setting manually change in DOM etc

useEffect() hook can achieve following life cycle feature of class component.

componentDidMount(), componentDidUpdate() and componentWillUnmount() combined.

Ex.

import React, { useState, useEffect } from "react";

import "./styles.css";

export default function App() {

  const [date, setDate] = useState("");

  useEffect(() => {

    const interval = setInterval(() => {

      const d = new Date();

      setDate(d.toLocaleTimeString());

    }, 1000);

    return () => {

      clearInterval(interval); // Cleanup on component unmount

    };

  },[]); // Runs only once after the component mounts

  return (

    <div className="App">

      <h1>Clock: {date}</h1>

    </div>

  );

}

**3.useContext**

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

Example:

**App.jsx**

import React from "react";

import ReactDom from "react-dom";

import D from "./D.jsx";

export UserContext=React.createContext();

let App=()=>{

return <><h1>I am App Component</h1>

<UserContext.Provider value={"Ganesh"}>

<D/>

</UserContext.Provider>

</>

}

export default App;

**3. Create Consumer**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Note: you have to import the context created by user

import React from "react";

import {UserContext} from "./App.jsx";

let F=(props)=>

{return <>

<UserContext.Consumer>

{

user=>{

return <h1>Contxt value is {user}</h1>

}

}

</UserContext.Consumer>

</>

}

export default F;

4. useRef

**5. useReducer**

useReducer() hook is alternative for useState() hook and generally more preferable by useState() means when we have complex logic with state variable then using useReducer() beneficial and when we have simple logic with state variable then we can use useState() hook

Ex.

import React, { useReducer } from "react";

import ReactDOM from "react-dom";

// Initial state

const initialCount = 0;

// Reducer function

const reducer = (state, action) => {

  switch (action) {

    case "add":

      return state + 1; // Increment the count

    case "sub":

      return state - 1; // Decrement the count

    case "reset":

      return 0; // Reset the count

    default:

      throw new Error("Unexpected Action");

  }

};

// Main App component

const App = () => {

  const [count, dispatch] = useReducer(reducer, initialCount); // useReducer hook

  return (

    <>

      <center>

        <h1>Initial Count is {count}</h1>

        <input

          type="button"

          value="Increment"

          onClick={() => dispatch("add")}

        />

        <br />

        <br />

        <input

          type="button"

          value="Decrement"

          onClick={() => dispatch("sub")}

        />

        <br />

        <br />

        <input type="button" value="Reset" onClick={() => dispatch("reset")} />

        <br />

        <br />

      </center>

    </>

  );

};

export default App;

**UseNavigate**

**useNavigate():** this hook introduce in react router v6 to replace useHistory() hook if we think previous version react router dom then we have useHistory() hook access the react router library and navigate to other routers

But after react-router dom v6 we have to use useNavigate() hook .

**App.jsx**

import React from "react";

import ReactDom from "react-dom";

import Home from "./Home.jsx";

import About from "./About.jsx";

import {BrowserRouter,Routes,Route} from "react-router-dom";

import {useNavigate} from "react-router-dom";

let App=()=>{

return <>

<BrowserRouter>

<Routes>

<Route path="/" element={<Home/>} />

<Route path="/about" element={<About/>} />

</Routes>

</BrowserRouter>

</>

}

export default App;

**About.jsx**

import React from "react";

import ReactDom from "react-dom";

import {useNavigate} from "react-router-dom";

let About=()=>{

let navigate=useNavigate();

return <>

<h1 style={{backgroundColor:"red",color:"white"}}>I am About component</h1>

<input type="button" name="s1" value="Go To Home" onClick={()=>navigate('/')} />

</>

}

export default About;

**Home.jsx**

import React from "react";

import ReactDom from "react-dom";

import {useNavigate} from "react-router-dom";

let Home=()=>{

let navigate=useNavigate();

return <>

<h1 style={{backgroundColor:"green",color:"white"}}>I am Home component</h1>

<input type="button" name="s" value="Go To About" onClick={()=>navigate('/about')} />

</>

}

export default Home;

1. **useMemo**

**usememo use for the avoid rendering**

**1. useMemo catches the return value of function and useCallback caches the function definition it self.**

**useMemo():** here memo indicate memorization means useMemo() hook is to get the memorized value of a function in react components and it works with concept of memorization which refer to catching the output of function for given arguments to save the computation time.

import React, { useState, useMemo } from "react";

function CounterComponent() {

  const [minus, setMinus] = useState(0);

  const [count, setCount] = useState(0);

  // Memoizing the multiplication result

  const multiplication = useMemo(() => count \* 10, [count]);

  return (

    <>

      <span>Multiplication: {multiplication}</span>

      <button onClick={() => setCount(count + 1)}>Increment</button>

      <span>Count: {count}</span>

      <button onClick={() => setMinus(minus - 1)}>Decrease</button>

      <span>Minus: {minus}</span>

    </>

  );

}

export default CounterComponent;

**7.useCallBack**

**useCallback caches the function definition it self.**

useCallback function help us to improve the performance of application called as useCallBack hook

**App.js**

import React,{useState,useMemo} from "react";

import ReactDom from "react-dom";

import ChildA from "./ChildA.jsx";

let App=()=>{

const [count,setCount]=useState(0);

let incCount=(e)=>{

setCount(count+1);

}

let learning=()=>{

//write here some logics

}

return<>

<ChildA Learning={learning}/>

<h1>Count is {count} </h1>

<input type='submit' name='s' value='Increment Counter' onClick={(e)=>incCount(e)}/>

</>

}

export default App;

**ChildA.jsx**

import React,{memo} from "react";

import ReactDom from "react-dom";

let ChildA=()=>{

console.log("I am child component");

return<></>

}

export default memo(ChildA) ;

8.useLayoutEffect

9.useEffectOnce

10.FormHooks

11.AnimationHooks

12. LayoutHooks

13. CustomHooks

1. **Difference of class and functional components**

|  |  |
| --- | --- |
| **Function component** | **Class component** |
| Function component is plain JavaScript function which accept props as parameter and return React JSX element | Class component is plain JavaScript class which accept properties from React.Component and override render() method and return React JSX element |
| Render() method not use by function component | In class component render() method must be override or define for return JSX element |
| Function component run from top to bottom and one the function component is returned it cannot be kept live | The class component instantiated and different life cycle method kept alive and is run and invoked depending as per phase of life cycle |
| Not required to use life cycle method in function component | Need to use life cycle method in function component |
| Hooks is facility in function component which help us to acquire the all life method feature as well as some additional feature like as state management in function component | Hooks not use in class component or class component not support to hooks concept |
| Constructor not used in function component | Constructor can use in class component |
| Function component known as stateless component if not use hooks | Class component known as stateful component |

**7. Pure component**

Pure component is component that enhance performance by implementing a shallow comparison of props in the state should component update life cycle method If there are no changes in the props or state,

**8. Higher order component**

A **Higher-Order Component (HOC)** in React is a design pattern used to enhance or extend the functionality of a component. It is a function that takes a component as an input and returns a new component with added features or logic.

Reverse Number

import React,{useState}from "react"

export default function ReverseApp(){

const [number,setnumber]=useState("");

const reversenumber=(num)=>{

if (num === "") return "";

return [...num.toString()].reverse().join("");

}

return(<div>

<input type="number" onChange={(e)=>{

const reverse = reversenumber(e.target.value);

setnumber(reverse);

}}/>

<h1>number:{number}</h1>

</div>)

}

**Capitalize string**

import React,{useState}from "react"

export default function ReverseApp(){

const [number,setnumber]=useState("");

const reversenumber=(str)=>{

if (str === "") return "";

return [...str.toUpperCase()].join("");

}

return(<div>

<input type="text" onChange={(e)=>{

const reverse = reversenumber(e.target.value);

setnumber(reverse);

}}/>

<h1>number:{number}</h1>

</div>)

}

**- Callback functions, promises, event loop**

**This keyword**

**In the class base react component this typically refer to the instance of the component itself.**

**By default js is not bind to this to class mentod you need to explicitly bind**

**Key Differences Between Class and Functional Components:**

| **Feature** | **Class Components** | **Functional Components** |
| --- | --- | --- |
| this usage | Required for accessing state, props, etc. | Not used, replaced by hooks. |
| State Management | Managed with this.state. | Managed with useState hook. |
| Methods Binding | Manual binding or arrow functions needed. | No binding required. |

**Ajax call**

**What is an AJAX Call?**

AJAX (**Asynchronous JavaScript and XML**) is a technique used to send and receive data from a server asynchronously without refreshing the web page. AJAX allows web applications to update content dynamically by exchanging data with a server in the background.

Modern AJAX calls often use JSON instead of XML, and libraries like fetch or axios are commonly used to simplify the process.

**How AJAX Works**

1. A browser sends an asynchronous request to a server using JavaScript.
2. The server processes the request and returns a response.
3. JavaScript handles the server's response and updates the webpage without reloading.

**Hoisting**

**Hoisting** is a JavaScript mechanism where variables, function declarations, and classes are moved to the top of their scope

**Higher order functions**

Higher-order functions are functions that can take one or more functions as arguments, return a function as a result, or both. In JavaScript (and many other programming languages), higher-order functions are an important concept for creating more reusable and modular code.

**Scope chain**

The **scope chain** in JavaScript refers to the series of scopes that the JavaScript engine traverses to look for variables when they are referenced in code

It is essential for variable lookup and determining how variables are accessed based on their location in the nested scopes.

**Global Scope**: The outermost scope, accessible throughout the entire program.

**Local Scope**: Any function or block scope where variables are only accessible within that function or block.

**Global execution context**

The **Global Execution Context (GEC)** is the default context in which any JavaScript code is executed when the program starts. It's the environment that is created when the script is first executed and serves as the base context for all other execution contexts, such as those created when functions are invoked.

**Key Features of the Global Execution Context:**

1. **Global Object**:
   * In a **browser**, the global object is the window object.
   * In **Node.js**, the global object is global.
   * This global object holds all global variables and functions.
2. **this Binding**:
   * In the global execution context, the this keyword refers to the global object (e.g., window in browsers or global in Node.js).
   * For example, this inside the global execution context will point to window in a browser.

Map function, filter function, reduce function

** map(): Transforms each element in the array and returns a new array.**

**const numbers = [1, 2, 3, 4];**

**const doubled = numbers.map(num => num \* 2);**

**console.log(doubled); // Output: [2, 4, 6, 8]**

** filter(): Filters elements based on a condition and returns a new array with only the elements that pass the condition.**

**const numbers = [1, 2, 3, 4];**

**const greaterThanTwo = numbers.filter(num => num > 2); console.log(greaterThanTwo); // Output: [3, 4]**

** reduce(): Reduces the array to a single value based on a given function.**

**const numbers = [1, 2, 3, 4]; const product = numbers.reduce((acc, num) => acc \* num, 1); console.log(product); // Output: 24**

Closure Topics:

**Promises**

Promises in React are widely used for handling asynchronous operations, such as fetching data from APIs, performing background tasks, or interacting with services.

**Note:** Before promises we need to know the synchronous operation and asynchronous operation

in JavaScript

**Synchronous operation:** synchronous operation means execute code line by line means

first operation successfully completed after that second operation get executed

Called as synchronous operation so by default java script is synchronous language and it is

Single threaded language.

**Asynchronous operation:** asynchronous operation means if first operation required some time for

Complete then within that time program can execute some other operation called as

**Note: Promises recommended in following terms.**

1. Producing code that can take some time

2. Consuming code that must wait for the result.

**Note:** JavaScript promises can be

**1. Pending:** while promise object is pending (working) the result is undefined.

**2. Fulfilled:** when promise object is fulfilled the result is value.

**3. Rejected:** when promise object is rejected the result is an error object.

**Promise**

A **Promise** in JavaScript is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

**Async await**

Async and Await is feature of JavaScript you can put your code on hold.

**async Functions**:

* Declares a function as asynchronous.
* Always returns a Promise, even if you return a non-Promise value.
* Inside an async function, you can use the await keyword.

**await Keyword**:

* Pauses the execution of an async function until the Promise resolves.
* Can only be used inside async functions.

javascript

Copy code

**Callback**

A **callback** is a function passed as an argument to another function.

callbacks are often used to handle asynchronous operations or to execute code after another operation has finished.

 **Asynchronous Programming**: Callbacks are heavily used in asynchronous programming, like handling events, making API calls, or executing tasks after a delay.

**Synchronous and Asynchronous**: Callbacks can be used in both synchronous and asynchronous contexts.

**Functions**

* Function is the block of reusable code designed to perform a specific task

Type of the function

1)Name function

2)Anonymous function

A function without a name, typically used as a value

3)Arrow function

It is the concise way to define function it is introduced in ES6. Arrow functiondo not have their own this.

**Higher-Order Functions**

A **higher-order function** is a function that:

1. Takes another function as an argument.
2. Returns a function as its result.

**Closure**

A **closure** is a feature in JavaScript where an inner function has access to the outer (enclosing) function's variables and scope, even after the outer function has finished execution.

Ex.

function outerFunction(outerVariable) {

return function innerFunction(innerVariable) {

console.log(`Outer: ${outerVariable}, Inner: ${innerVariable}`);

};

}

const closureFunction = outerFunction("Outer Scope");

closureFunction("Inner Scope");

**Currying**

**Currying in JavaScript**

**Currying** is a functional programming technique in JavaScript where a function takes multiple arguments, but instead of accepting them all at once,

Ex.

function add(a, b) {

return a + b;

}

console.log(add(5, 3)); // Output: 8

Question:

String and arrays PE hi hote hai

Ex.

**Reverse string,**

1)import React,{useState}from "react"

export default function Reverse(){

const [stringd,setstring]=useState("");

const reverses=(str)=>{

return [...str].reverse().join("");

}

return(<>

<input type="text" onChange={(e)=>{

const reversestring = reverses(e.target.value);

setstring(reversestring);

}}/>

<h1>reverse string is : {stringd}</h1>

</>)

}

2)

import React, { useState } from "react";

export default function Reverse() {

const [stringd, setstring] = useState("");

const reverses = (str) => {

return [...str].reverse().join("");

};

// Predefined string

const str = "pravin";

// Reverse the string immediately using your logic

const reversedString = reverses(str);

setstring(reversedString);

return ( <>

<h1>Original String: {str}</h1>

<h1>Reversed String: {stringd}</h1>

</>

);

}

Palindrome

Str = “madam”;

import React, { useState } from "react";

export default function Reverse() {

const [stringd, setstring] = useState("");

const reverses = (str) => {

return [...str].reverse().join("");

};

// Predefined string

const str = "pravin";

// Reverse the string immediately using your logic

const reversedString = reverses(str);

setstring(reversedString);

return (

<>

<h1>Original String: {str}</h1>

<h1>Reversed String: {stringd}</h1>

</>

);

}

String Palindrome or Not check

import React from "react";

export default function Palindrome() {

const str = "pravin madhavrao gorade"; // Hardcoded string

const isPalindrome = (str) => {

const reversedString = [...str].reverse().join("");

return str === reversedString; // Compare original with reversed

};

const result = isPalindrome(str);

return (

<>

<h1>

The string "{str}" is {result ? "a palindrome!" : "not a palindrome."}

</h1>

</>

);

}

Missing number in array

import React from "react"

export default function Palindrome(){

const a = [1,3,5,6,7,10];

let missing = [];

let current = 1;

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

{

while(current<a[i])

{

missing.push(current);

current++;

}

current++;

}

return(

<h1>missing number:{missing.join(",")}</h1>

)

}

Second highest number in array

import React from "react"

export default function SecondMax(){

const a = [1,3,5,6,7,10];

let missing = [];

let min = 0;

let smin = 0;

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

{

if(a[i]>min)

{

min=a[i];

}

}

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

{

if(a[i]>smin && a[i]<min)

{

smin=a[i];

}

}

missing.push(smin);

return(

<h1>missing number:{missing.join(",")}</h1>

)

}

Or

import React from "react";

import "./styles.css";

export default function App() {

let arr = [10, 20, 30, 40, 50,60,60];

let sm = Math.max(...arr);

let secondm = Math.max(...arr.filter((each) => each !== sm));

return (

<div className="App">

<h1>second max:{secondm}</h1>

</div>

);

}

Duplicate character in array

import React from "react";

import "./styles.css";

export default function App() {

let arr = ["a", "b", "c", "a", "d", "e", "b", "f"];

// Find duplicates using filter and indexOf

let duplicates = arr.filter((item, index) => arr.indexOf(item) !== index);

// Remove duplicates from the duplicates array using Set

duplicates = [...new Set(duplicates)];

return (

<div className="App">

<h1>Duplicate Characters: {duplicates.join(", ")}</h1>

</div>

);

}

Yeh react mein puchte hai

-What is the Output of following code:

Promise.resolve(3)

.then((res)=>{

console.log(res)

})

.catch()

.then()

.then((res)=>{

console.log(res)

})

Output:**-3 and undefine**

function outer(){

var b = 2

function inner(){

b++;

console.log(b);

//var b = 3;

}

inner();

}

outer();

Output nun

-Write code for the pattern

n=7

1 2 3 4 5 \* \*

n=17

1 2 3 4 5 \* \* \* \* \*

11 12 13 14 15 \* \*

n=27

1 2 3 4 5 \* \* \* \* \*

11 12 13 14 15 \* \* \* \*\*

21 22 23 24 25 \* \*

->public class ComponeyPattern {

public static void main(String[] args) {

int n = 27; // Change this value to 7, 17, or any other number.

int current = 1;

// Continue until we reach 'n'

while (current <= n) {

// Print the first 5 numbers in the row

for (int i = 0; i < 5 && current <= n; i++) {

System.out.print(current + " ");

current++;

}

// Print stars for the rest of the row

for (int i = 0; i < 5 && current <= n; i++) {

System.out.print("\* ");

current++;

}

System.out.println(); // Move to the next row

}

}

}

-Write a program reverse the integer without using any built in method

import React from "react"

export default function REVerse(){

let no = 123;

let rem=0;

let sum =0;

while(no!=0)

{

rem = no%10;

no = (no-(no % 10))/10;

sum = sum \* 10 + rem;

}

return(

<h1>Reverse:{sum}</h1>

)

}

-Write a program to find the second largest element of array

import React from "react"

export default function SecondMax(){

const a = [1,3,5,6,7,10,14,39];

let missing = [];

let min = 0;

let smin = 0;

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

{

if(a[i]>min)

{

min=a[i];

}

}

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

{

if(a[i]>smin && a[i]<min)

{

smin=a[i];

}

}

missing.push(smin);

return(

<h1>Second hieghts:{missing.join(",")}</h1>

)

}

Dom vs virtual Dom JavaScript tricky hai do prepared

| **Feature** | **DOM (Document Object Model)** | **Virtual DOM** |
| --- | --- | --- |
| **Definition** | The actual structure that represents the HTML elements in the browser. | A lightweight, in-memory representation of the real DOM. |
| **Performance** | Slower when it comes to updating the UI, especially with large changes, as it directly manipulates the DOM tree. | Faster because it minimizes updates to the real DOM by performing diffs and batching updates. |
| **Size** | Large, as it represents the entire UI structure of the page. | Lightweight and smaller because it doesn't render the entire UI. |
| **Manipulation** | DOM manipulation involves reflow and repaint, which can be expensive. | Virtual DOM updates only the necessary parts of the real DOM after efficient diffing. |
| **Usage** | Used directly in traditional JavaScript frameworks and vanilla JS. | Used in frameworks like React, Vue.js, etc., to optimize UI updates. |
| **Reconciliation** | No built-in reconciliation mechanism; updates directly affect the DOM. | Uses a diffing algorithm to determine the minimal set of changes needed to update the real DOM. |
| **Efficiency** | Inefficient for frequent UI updates, leading to performance bottlenecks. | Efficient for frequent UI updates due to minimal real DOM manipulation. |
| **Real-world Analogy** | A physical book that you directly edit. | A draft copy of the book where changes are made before finalizing in the physical copy. |

**Why Virtual DOM?**

* Reduces **expensive real DOM operations**.
* Optimizes UI rendering and improves performance.

**What is a DOM Tree?**

* The **DOM Tree** is a hierarchical representation of the structure of a web document

Pattern

let row = 5;

let col = 9;

for(let i=1;i<=row;i++)

{

let count = ' ';

for(let j=1;j<=col;j++)

{

if(j<=4+i && j>=6-i)

{

count+='\*';

}

else{

count+=' ';

}

}

console.log(count);

}

Output

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*