Time : 1 hr (60 min)

Note:

|  |
| --- |
| * All questions need to answer. * Examination should be completed in given amount of time * Candidate can write any logic which suitable for coding questions for Javascript and react * They should write code in Javascript or react * Answer sheets will be evaluated by our team members. |

**Questions (JavaScript)**

1. What is hoisting in JavaScript, what happens with var, let and const during Hoisting?

Ans:Hoisting in javaScript is a behavior where variable and function declarations are moved to the top of their scope during the compilation phase, even before the code is executed. This means you can use variables or functions before they are actually declared in the code.

**Var-** When a variable is declared using var, it is hoisted to the top of its scope and initialized with undefined. This is why you can access a var variable before its declaration, but it will return undefined.

For example:

console.log(x); // undefined

var x = 5;

console.log(x); // 5

**let **and** const :**Variables declared with let and const are also hoisted, but they are not initialized. Instead, they remain in a "temporal dead zone" from the start of the block until the declaration is encountered. If you try to access them before their declaration, you’ll get a ReferenceError.

For example:

console.log(y); // ReferenceError

console.log(y); // 10

1. Problem: What is the output of the following code?

1. console.log(x)

2. var x = 5

3.

4. console.log(y)

5. let y = 5

6.

7. console.log(z)

8. const z = 5

Output:

undefined

ReferenceError

ReferenceError

1. What is the difference between == and === in JavaScript? What will be the output of the below statements?

This == operator compares the values of two variables after performing type conversion if necessary ,this strict equility === operator compares the values of two variables without performing type conversion

1. console.log(1 == '1') // output = true

2. console.log(1 === '1') // output = false

3. console.log(1.5 == '1.5') // output = true

4. console.log((0.1 + 0.2) == 0.3) // output = true

5. console.log(0 == false) // output = true

6. console.log(0 === false) // output = false

7. cosmole.log(true == !0) // output = true

8. console.log(1 === 1 ? 'yes' : 'no') // output = yes

1. What is callback hell and explain with the example?

Ans: Callback hell in JavaScript happens when there are too many nested callback functions, making the code messy and hard to understand. It usually occurs during asynchronous tasks like API calls etc. As more callbacks are added, the code forms a "pyramid" shape with lots of indentation, making it difficult to read, debug, or update.

Example:

firstTask(function(result1) {

secondTask(result1, function(result2) {

thirdTask(result2, function(result3) {

console.log('final result: ', result3);

});

});

});

1. What is difference between callback and promises and async/await? Give example of async/await function call.

Ans: **Callback**: Function passed as an argument and executed later, can lead to messy code if too many are used.

**Promise**: Represents a value that will be resolved later, helps avoid callback hell by chaining .then() and .catch().

**Async/Await**: Makes asynchronous code look like synchronous code, using async for the function and await to pause execution until the promise is resolved.

Example of async/await :

async function fetchData() {

return new Promise((resolve) => {

setTimeout(() => {

resolve("Data fetched");

}, 1000);

});

}

async function getData() {

const result = await fetchData();

console.log(result); // Output: Data fetched

}

getData();

1. What is the output of the following code snippet and explain why?

1. const str = 'javascript'

2. str.replace('javascript', 'javascript programming')

3. console.log(str) // output : javascript

Why?answer:The replace() method returns a new string but does not modify the original string unless you explicitly assign it to a variable

1. What is the difference in call, bind and apply in JavaScript? Also solve the following:

Ans: The call() method is used to call a function with a specified this value and arguments provided individually .

The apply() method is similar to call(), but instead of passing arguments one by one, it takes an array of arguments.

The bind() method creates a new function that, when called, has its this value set to the provided object.

a.When Should you use **call** instead of apply?

Ans:**Use** call() when you want to pass arguments **individually**.

b. how does **bind** differ from call and apply?

Ans:call() **and** apply() invoke the function **immediately** with the provided this value and arguments.

bind(), on the other hand, **does not invoke** the function immediately. Instead, it **returns a new function** that can be called later with a specific this value.bind() is useful when you want to set the this context for a function and call it later.

C.Provide a scenario where **bind** is necessary but **call**/**apply** are not.

Ans:A scenario where bind() is necessary but call() or apply() are not suitable is when you're setting up event handlers or callbacks, where the function needs to be invoked later, not immediately.

1. What is closure? Create a function createCounter (Use closures to maintain the state of the counter) that returns an object with two methods: increment and getValue. The increment method should increase the counter by 1 each time it is called, and the getValue method should return the current value of the counter.

Ans:A **closure** is a function that remembers the variables from its outer scope even after that scope has exited. Closures are created every time a function is defined, allowing the inner function to access and remember variables from its parent function.

Example:

function createCounter() {

let counter = 0;

return {

increment: () => {

counter++;

},

// Get the current value of the counter

getValue: () => {

return counter;

}

};

}

const counter1 = createCounter();

counter1.increment();

counter1.increment();

console.log(counter1.getValue()); // Output: 2

const counter2 = createCounter();

counter2.increment();

console.log(counter2.getValue()); // Output: 1

1. Write a program to remove odd numbers from array without creating a new array object.
   1. For loop/Iterators and intermediate variables can be used.
   2. NOTE: DO NOT assign a new object/modified object to inputArray again, odd numbers must be removed from original inputArray object
   3. Example – inputArray = [4, 3, 6, 7, 11, 45, 16, 19, 56, 99]

Output of console.log(inputArray) should be [4, 6, 16, 56]

Ans:

Example:

function removeOddNumbers(inputArray) {

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

// Check if the current element is odd

if (inputArray[i] % 2 !== 0) {

// Remove the odd element

inputArray.splice(i, 1);

// Decrement the index to account for the shifted elements

i--;

}

}

}

// Example Usage

const inputArray = [4, 3, 6, 7, 11, 45, 16, 19, 56, 99];

removeOddNumbers(inputArray);

console.log(inputArray); // Output: [4, 6, 16, 56]

**Questions (React)**

1. What is Virtual Dom in React?

Ans: The **Virtual DOM (VDOM)** in React is a lightweight, in-memory representation of the **real DOM** that helps React efficiently update and render web applications. Instead of directly manipulating the real DOM, React uses the Virtual DOM to optimize performance.

1. Difference between UseCallback, UseMemo and React.memo in React?

Ans:

**useCallback**: A React Hook that memoizes a function, preventing its recreation on every render unless its dependencies change. Used to optimize performance when passing functions as props.

**useMemo**: A React Hook that memoizes the result of a computation, avoiding its recalculation on every render unless its dependencies change. Used to optimize expensive calculations.

**React.memo**: A higher-order component that memoizes a functional component, preventing it from re rendering unless its props change. Used to optimize components with stable props.

1. What is the purpose of Key in React Lists, and why should not use index as key?

Ans:key is a special attribute used to identify elements in a list and help React efficiently update and manage the list when its items change. It acts as a unique identifier for each list item, allowing React to track the elements between renders.

1. What happens when you return null from a React component?

1. const App = ({ shouldRender }) => {

2.

3. if (!shouldRender) {

4. return null

5. }

6.

7. return <div>App</div>

8. }

Ans: When you return null from a React component, React will **render nothing** for that component. Essentially, it tells React to **skip rendering the component** and **not display anything in the DOM** for that component.

1. What will be the output of the following component when we click on “Increment” button?

1. function Counter() {

2. const [count, setCount] = useState(0)

3.

4. function handleClick() {

5. setCount(count + 1)

6. setCount(count + 1)

7. }

8.

9. return (

10. <div>

11. <p>{count}</p>

12. <button onClick={handleClick}>Increment</button>

13. </div>

14. )

15. }

Ans: React will always update the count based on the previous state, and you'll get the expected value of **2** after the button is clicked.

1. What are Higher Order Components (HOCs) in React?

Ans:A **Higher-Order Component (HOC)** is a function that takes a component and returns a **new component** with enhanced functionality. In simpler terms, an HOC is a pattern that allows you to reuse component logic.

HOCs do not modify the original component, instead, they wrap it with additional logic or behavior.

1. What is the difference between controlled and uncontrolled components in React?

| **Feature** | **Controlled Component** | **Uncontrolled Component** |
| --- | --- | --- |
| **State Management** | Form data is managed by React state. | Form data is managed by the DOM. |
| **Form Value** | The value is passed via value prop and updated by React. | The value is accessed via ref and handled by the DOM. |
| **Event Handling** | Event handlers are used to update state. | No need for event handlers to update the value. |
| **Complexity** | More complex as React manages the state for every input. | Simpler but less flexible; React doesn't manage input state. |
| **Use Case** | When you need to interact with or validate form data in real time. | When you don’t need to track form data or need quick implementation. |