Javascript

Theory:

1. What do you understand about JavaScript?

JavaScript is a **programming language** mainly used to add interactivity to websites. It helps make things dynamic like pop-ups, sliders, form validation, and even full web applications.

2. What's the difference between JavaScript and Java?

Even though their names are similar, they're completely different:

- **Java** is a compiled, object-oriented programming language used for apps and backend systems.
- **JavaScript** is mostly used for frontend web development and runs directly in the browser.
  They serve different purposes.

3. What are the various data types that exist in JavaScript?

JavaScript has the following data types:

- **String** (text)
- **Number**
- **Boolean** (true/false)
- **Undefined**
- **Null**
- **Object**
- **Symbol**
- **BigInt** (for very large numbers)

4. What are the features of JavaScript?

Some key features:

- Lightweight and easy to use
- Can run directly in the browser
- Supports event handling (like button clicks)
- Works well with HTML and CSS
- Supports both object-oriented and functional programming

5. What are the advantages of JavaScript over other web technologies?

- Runs fast in the browser

- Doesn’t need to reload the page for updates (using AJAX)
- Big community and support
- Can be used both on frontend (with HTML/CSS) and backend (with Node.js)

6. How do you create an object in JavaScript?
You can create an object like this:

```
const person = {
  name: "Alice",
  age: 25,
  isStudent: false
};
```

7. How do you create an array in JavaScript?
Here’s how you create an array:

```
const colors = ["red", "green", "blue"];
```

8. What are some of the built-in methods in JavaScript?
JavaScript provides many built-in methods. A few examples:

- `Math.random()` – gives a random number
- `toUpperCase()` – converts text to uppercase
- `push()` – adds an item to an array
- `pop()` – removes the last item from an array

9. What are the scopes of a variable in JavaScript?
There are mainly 3 scopes:

- **Global scope** – accessible everywhere
- **Function scope** – available only inside a function
- **Block scope** – limited to `{}` blocks, like in `if`, `for`, etc. (with `let` and `const`)

10. What is the ‘this’ keyword in JavaScript?
`this` refers to the **current object** that is being used.

- Inside an object method, `this` refers to that object.
- In global scope, it refers to the window object (in browsers).
  Its behavior can change depending on where it's used.

11. What are the conventions of naming a variable in JavaScript?
Some basic rules:

- Can contain letters, digits, `$`, and `_`
- Can't start with a number
- Usually written in **camelCase** like `userName`
- Should be meaningful and clear

12. What is Callback in JavaScript?
A **callback** is a function passed into another function as an argument. It runs after the first function finishes.
Example:

```
function greet(name, callback) {
  console.log("Hi " + name);
  callback();
}
```

13. How do you debug a JavaScript code?
Common ways to debug:

- Use `console.log()` to see values
- Use browser **Developer Tools** (like Chrome DevTools)
- Set breakpoints and step through the code
- Use `debugger;` statement to pause code in DevTools

14. What is the difference between Function declaration and Function expression?

Function declaration is hoisted (can be used before it's defined):

```
function sayHi() { }
```

-

Function expression is not hoisted:

```
const sayHi = function() { }
```

-

15. What are the ways of adding JavaScript code in an HTML file?
There are 3 ways:

- **Inline** – inside an element: `<button onclick="alert('Hi')">Click</button>`
- **Internal** – inside a `<script>` tag in the HTML file
- **External** – using a separate `.js` file linked with `<script src="script.js"></script>`

16. What do you understand about cookies?
Cookies are small pieces of data stored in the browser. Websites use them to remember things like login info, user preferences, or items in a shopping cart.

17. How would you create a cookie?
You can create a cookie using JavaScript like this:

```
document.cookie = "username=John; expires=Fri, 25 Jul 2025 12:00:00 UTC; path=/";
```

18. How would you read a cookie?
To read a cookie, you just access:

```
document.cookie
```

It returns a string of all cookies for that page.

19. How would you delete a cookie?
To delete a cookie, set its expiry date in the past:

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";
```

20. What's the difference between `let` and `var`?

- `var` is function-scoped
- `let` is block-scoped (inside `{ }`)
  Also, `let` doesn't allow redeclaration in the same block, while `var` does.

21. What are Closures in JavaScript?
A **closure** is when a function remembers the variables from its parent scope even after that parent function has finished running.
Example:

```
function outer() {
  let count = 0;
  return function inner() {
    count++;
    console.log(count);
  };
}
```

22. What are the arrow functions in JavaScript?
Arrow functions are a shorter way to write functions.
Example:

```
const add = (a, b) => a + b;
```

They also don't have their own `this`—they use the `this` from the outer scope.

23. What are the different ways an HTML element can be accessed in a JavaScript code?
You can access elements using:

- `getElementById()`
- `getElementsByClassName()`
- `getElementsByTagName()`
- `querySelector()`
- `querySelectorAll()`

24. What are the ways of defining a variable in JavaScript?
You can define variables using:

- `var` (older, function-scoped)
- `let` (modern, block-scoped)
- `const` (block-scoped and can't be reassigned)

25. What are Imports and Exports in JavaScript?
They allow us to split code across files:

- **export** – makes a function or variable available to other files
- **import** – brings that exported code into a file
  Example:

```
// file1.js
export const name = "John";

// file2.js
import { name } from "./file1.js";
```

26. What is the difference between Document and Window in JavaScript?

- `window` represents the **browser window**
- `document` is the **webpage (DOM)** loaded inside that window
  So `document` is a part of `window`

27. What are some of the JavaScript frameworks and their uses?

- **React** – for building user interfaces
- **Vue** – a lightweight frontend framework
- **Angular** – a full-featured frontend framework
- **Node.js** – for running JavaScript on the server side

28. What is the difference between Undefined and Undeclared in JavaScript?

- **Undefined** means a variable was declared but no value was given
- **Undeclared** means the variable was never declared at all

29. What is the difference between Undefined and Null in JavaScript?

- `undefined` means no value has been assigned
- `null` means a value is intentionally empty or "nothing"

30. What is the difference between Session storage and Local storage?

- **Session storage** stores data for one tab and gets cleared when the tab is closed
- **Local storage** stores data permanently until you manually clear it
  Both store key-value pairs and don't expire by themselves.

Problem :

1. Find the largest number in an array

```
const numbers = [10, 25, 8, 99, 4];
const max = Math.max(...numbers);
console.log(max);
```

Explanation:
The spread operator (`...`) unpacks array elements, and `Math.max()` returns the largest one.

---

2. Remove duplicates from an array

```
const numbers = [1, 2, 2, 3, 4, 4, 5];
const unique = [...new Set(numbers)];
console.log(unique);
```

Explanation:
`Set` only stores unique values. Using spread operator, we convert it back to an array.

3. Check if a value exists in an array

```
const fruits = ["apple", "banana", "orange"];
console.log(fruits.includes("banana")); // true
```

Explanation:
`includes()` checks whether a given value is present in the array.

4. Sum all values in an array

```
const nums = [5, 10, 15];
const sum = nums.reduce((total, num) => total + num, 0);
console.log(sum); // 30
```

Explanation:
`reduce()` adds all elements of the array starting from 0.

5. Reverse an array without modifying the original

```
const arr = [1, 2, 3];
const reversed = [...arr].reverse();
console.log(reversed); // [3, 2, 1]
console.log(arr); // [1, 2, 3]
```

Explanation:
We copy the array using spread operator and then reverse it, so the original stays unchanged.

6. Count item occurrences in an array

```
const items = ["apple", "banana", "apple", "orange", "banana"];
const count = {};

items.forEach(item => {
  count[item] = (count[item] || 0) + 1;
});

console.log(count);
// { apple: 2, banana: 2, orange: 1 }
```

Explanation:
We loop through each item and use an object to store how many times each item appears.

7. Merge two objects

```
const obj1 = { a: 1, b: 2 };
const obj2 = { b: 3, c: 4 };
const merged = { ...obj1, ...obj2 };
console.log(merged); // { a: 1, b: 3, c: 4 }
```

Explanation:
Using the spread operator, we merge both objects. If keys match, the second one overrides the first.

8. Find an object from an array of objects

```
const users = [
  { id: 1, name: "Alice" },
  { id: 2, name: "Bob" }
];
const user = users.find(u => u.id === 2);
console.log(user); // { id: 2, name: "Bob" }
```

Explanation:
The `find()` method returns the first object that matches the condition (`id === 2`).

9. Update a value in an object inside an array

```
const users = [
  { id: 1, name: "Alice" },
  { id: 2, name: "Bob" }
];
const updated = users.map(user =>
  user.id === 2 ? { ...user, name: "Robert" } : user
);
console.log(updated);
```

Explanation:
We use `map()` to go through each object. If the `id` matches, we return a new object with the updated name.

10. Convert an object into an array of key-value pairs

```
const person = { name: "John", age: 30 };
const entries = Object.entries(person);
console.log(entries);
// [["name", "John"], ["age", 30]]
```

Explanation:
`Object.entries()` turns an object into an array of `[key, value]` pairs.

React.js

1. What are the features of React?

- **Component-based**: You can break the UI into reusable pieces.
- **Virtual DOM**: It makes React fast by updating only the necessary parts of the page.
- **Unidirectional data flow**: Data flows in one direction, making things easier to debug.
- **JSX support**: Lets you write HTML inside JavaScript.
- **Fast and efficient**: Because of how it handles DOM updates and rendering.

2. What is JSX?

JSX stands for **JavaScript XML**. It lets you write HTML-like code inside JavaScript.
For example:

```
const element = <h1>Hello React</h1>;
```

It makes writing UI in React feel easier and cleaner.

3. Can web browsers read JSX directly?

No, browsers **can't understand JSX**. Tools like **Babel** are used to convert JSX into regular JavaScript so the browser can read it.

4. What is the virtual DOM?

The virtual DOM is a **copy of the real DOM** that React uses in memory. When something changes, React updates the virtual DOM first, compares it with the old one, and then only updates the changed part in the real DOM. This makes things faster and smoother.

5. Why use React instead of other frameworks, like Angular?

- React is **lighter and easier to learn**
- It focuses only on the **UI layer**, so it's more flexible
- Uses the **virtual DOM**, which improves performance
- Has a **large community and lots of support**
- Easier to reuse code through components

6. What is the difference between the ES6 and ES5 standards?

ES5 is the older version of JavaScript, while **ES6** (also called ES2015) brought many new features like:

- `let` and `const` (instead of `var`)
- Arrow functions (`=>`)
- Template literals (`` `Hello ${name}` ``)
- Classes
- Destructuring
- Modules (`import` and `export`)
  ES6 makes code **shorter, cleaner, and easier to manage**.

7. What is an event in React?

An event in React is an action that happens in the browser, like a click, a key press, a form submit, etc. React lets us handle these events using functions.

8. How do you create an event in React?

You create events using camelCase syntax and pass a function.
Example:

```
<button onClick={handleClick}>Click me</button>
```

Here, `handleClick` is called when the button is clicked.

9. What are synthetic events in React?

Synthetic events are React's way of handling browser events. They work the same as native events but behave the same across all browsers. React wraps native events inside its own synthetic event system.

10. Explain how lists work in React.

In React, lists are created by using the `.map()` function to loop through an array and return JSX for each item.
Example:

```
const items = ["Apple", "Banana"];

const listItems = items.map(item => <li>{item}</li>);
```

11. Why is there a need to use keys in Lists?

Keys help React identify which items have changed, added, or removed. It makes list rendering more efficient. Each list item should have a unique key.

12. What are forms in React?

Forms in React are just like HTML forms, but React handles the values and updates using state. This makes the form more interactive and controlled.

13. How do you create forms in React?

You use `<form>` with input fields, and store the values in state. Use `onChange` to update the state.
Example:

```
<input value={name} onChange={(e) => setName(e.target.value)} />
```

14. How do you write comments in React?

- In JSX: `{/* This is a comment */}`
- In normal JavaScript: `// comment` or `/* comment */`

15. What is an arrow function and how is it used in React?

Arrow functions are a short way to write functions. They're often used in React for event handlers or inside components.
Example:

```
const handleClick = () => { ... }
```

16. How is React different from React Native?

React is used to build websites. React Native is used to build mobile apps for Android and iOS. They use the same logic, but the UI components are different.

17. How is React different from Angular?

React is a library that only handles the UI. Angular is a full framework that includes routing, HTTP, etc. React is more flexible, and Angular is more structured.

18. What are React Hooks?

Hooks are special functions in React that let you use features like state, effects, and context in functional components. Examples include `useState`, `useEffect`, and `useContext`.

19. What is useState, and how does it work?

`useState` is a hook that allows you to create and update state in a functional component.
Example:

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

20. What is useEffect, and how does it differ from lifecycle methods in class components?

`useEffect` is used to run side effects like fetching data or setting a timer. In class components, we used `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`. `useEffect` replaces all of them.

21. What is Memoization in React?

Memoization means caching the result of a function or component to avoid unnecessary re-renders. `React.memo` and `useMemo` are used for this in React.

22. What is Prop Drilling and how do you avoid it?

Prop drilling happens when you pass data through many components just to reach a child. You can avoid it by using **Context API** or **state management libraries** like Redux or Zustand.