**WEEK 7**

**9.ReactJS-HOL**

**1. List the Features of ES6 (ECMAScript 2015)**

ES6 introduced many powerful features to JavaScript. Key features include:

1. let and const – Block-scoped variable declarations
2. Arrow Functions – Shorter function syntax with lexical this
3. Classes – Object-oriented class-based syntax
4. Template Literals – ${} interpolation inside backticks
5. Destructuring – Easy unpacking of arrays and objects
6. Default Parameters – Assign default values to function parameters
7. Rest and Spread Operators (...) – Gather or expand elements
8. Modules – import and export for modular code
9. Promises – Better asynchronous code handling
10. Map and Set – New collection data structures
11. for...of Loop – Iterates over iterable objects
12. Symbol – Unique and immutable primitive values
13. Enhanced Object Literals – Shorthand for properties and methods

**2. Explain JavaScript let**

Definition:  
let is a keyword introduced in ES6 to declare variables that are block-scoped, meaning they are only accessible within the {} where they are defined.

Example:

let x = 10;

if (true) {

let x = 20; // different x, scoped to this block

console.log(x); // 20

}

console.log(x); // 10

Key Features:

* Block-scoped
* Can be updated, but not re-declared in the same scope
* Helps avoid variable leakage outside of loops or blocks

**3. Identify the Differences Between var and let**

|  |  |  |
| --- | --- | --- |
| Feature | var | let |
| Scope | Function-scoped | Block-scoped |
| Hoisting | Hoisted and initialized as undefined | Hoisted but not initialized |
| Redeclaration | Allowed in same scope | Not allowed in same scope |
| Global Property | Becomes part of window object | Does not attach to window |

Example:

var a = 5;

var a = 10; // Allowed

let b = 5;

// let b = 10; // Error: already declared

**4. Explain JavaScript const**

Definition:  
const is used to declare constants—variables whose value cannot be reassigned.

Example:

const PI = 3.14;

// PI = 3.1415; // Error

Important Points:

* const is block-scoped (like let)
* You must assign a value during declaration
* The reference can't be changed, but for objects/arrays, contents can still be modified

const arr = [1, 2, 3];

arr.push(4); // ( allowed)

// arr = [5,6]; // ( not allowed)

**5. Explain ES6 Class Fundamentals**

Definition:  
ES6 introduced the class syntax to define objects and handle inheritance, making JavaScript more object-oriented and readable.

Syntax:

class Person {

constructor(name) {

this.name = name;

}

greet() {

console.log(`Hello, ${this.name}`);

}

}

const p1 = new Person("Harshika");

p1.greet(); // Hello, Harshika

Key Concepts:

* Use constructor() for initialization
* Use methods without function keyword
* Classes are not hoisted

**6. Explain ES6 Class Inheritance**

Definition:  
ES6 supports inheritance using the extends keyword, allowing one class to inherit properties and methods from another.

Example:

class Animal {

constructor(name) {

this.name = name;

}

speak() {

console.log(`${this.name} makes a sound`);

}

}

class Dog extends Animal {

speak() {

console.log(`${this.name} barks`);

}

}

const d = new Dog("Bruno");

d.speak(); // Bruno barks

Key Concepts:

* Use extends to inherit
* Use super() to call parent constructor
* You can override methods

**7. Define ES6 Arrow Functions**

Definition:  
Arrow functions are a concise way to write function expressions using =>. They do not have their own this, arguments, or super.

Syntax:

// Traditional

function add(a, b) {

return a + b;

}

// Arrow function

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

Features:

* Shorter syntax
* Implicit return (if one expression, no {} needed)
* Lexical this binding — useful in callbacks

Example with implicit return:

const square = x => x \* x;

**8. Identify set() and map()**

➤ Set

* A collection of unique values.
* No duplicates allowed.
* Values can be of any type.

const mySet = new Set([1, 2, 2, 3]);

console.log(mySet); // Set(3) {1, 2, 3}

mySet.add(4);

mySet.has(2); // true

➤ Map

* A collection of key-value pairs.
* Keys can be any type (not just strings).
* Preserves insertion order.

const myMap = new Map();

myMap.set('name', 'Harshika');

myMap.set(1, 'one');

console.log(myMap.get('name')); // Harshika

|  |  |  |
| --- | --- | --- |
| Feature | Set | Map |
| Stores | Unique values | Key-value pairs |
| Key type | Only values | Any data type as key |
| Order | Maintains insertion order | Maintains insertion order |

**10.ReactJS-HOL**

**1. Define JSX**

Definition:  
JSX (JavaScript XML) is a syntax extension for JavaScript used with React to describe what the UI should look like. JSX looks similar to HTML but is actually syntactic sugar for React.createElement() calls.

Example:

const element = <h1>Hello, world!</h1>;

Key Points:

* JSX allows embedding HTML-like code in JavaScript.
* JSX is not required in React, but it makes the code more readable.
* JSX is compiled to JavaScript using tools like Babel.

JSX makes writing UI code easier, more intuitive, and closer to HTML.

**2. Explain about ECMAScript**

Definition:  
ECMAScript (ES) is the standardized scripting language specification on which JavaScript is based. It defines how the language behaves.

Popular Versions:

* ES5: Introduced strict mode, Array.forEach, etc.
* ES6 (ES2015): Introduced arrow functions, let, const, classes, modules, template literals, etc.
* Later versions: Added features like optional chaining, nullish coalescing, and more.

Why it's important in React:

React uses modern JavaScript features (mostly from ES6 and beyond) such as:

* Arrow functions (() => {})
* Classes
* import/export syntax
* Destructuring
* Spread/rest operators

Knowing ECMAScript features helps write cleaner and more modern React code**.**

**3. Explain React.createElement()**

Definition:  
React.createElement() is a core method used internally by JSX to create a React element (a virtual DOM object).

Syntax:

React.createElement(type, props, ...children)

Example Without JSX:

const element = React.createElement('h1', { className: 'greeting' }, 'Hello');

Equivalent JSX:

const element = <h1 className="greeting">Hello</h1>;

JSX is just a cleaner and more readable way of writing React.createElement().

**4. Explain How to Create React Nodes with JSX**

Definition:  
React nodes are UI elements (like <div>, <h1>, <p>) created using JSX that get compiled into JavaScript objects representing elements in the virtual DOM.

How to create them:

const header = <h1>This is a heading</h1>;

const paragraph = <p>This is a paragraph</p>;

* Nodes can also be composed using variables and functions.
* They can include children, props, and event handlers.

Example of nested JSX:

const element = (

<div>

<h1>Hello!</h1>

<p>Welcome to React</p>

</div>

);

These nodes are later rendered to the actual DOM using React DOM APIs.

**5. Define How to Render JSX to the DOM**

Definition:  
To render JSX to the browser DOM, we use the ReactDOM.render() (React 17 and below) or createRoot().render() in React 18+.

React 17 and below:

ReactDOM.render(<App />, document.getElementById('root'));

React 18+ syntax:

import { createRoot } from 'react-dom/client';

const root = createRoot(document.getElementById('root'));

root.render(<App />);

This process converts virtual DOM elements into actual browser DOM elements.

**6. Explain How to Use JavaScript Expressions in JSX**

Definition:  
You can embed JavaScript expressions (like variables, functions, or operations) inside JSX using curly braces {}.

Examples:

const name = 'Harshika';

const element = <h1>Hello, {name}!</h1>;

* Valid expressions include variables, function calls, conditionals (? :), math, etc.
* Statements like if, for cannot be used directly inside JSX.

<p>2 + 2 = {2 + 2}</p> // Output: 2 + 2 = 4

Embedding JavaScript expressions allows dynamic content in UI.

**7. Explain How to Use Inline CSS in JSX**

Definition:  
Inline styling in JSX is done by assigning a JavaScript object to the style attribute. The property names follow camelCase (not hyphenated).

Syntax:

const headingStyle = {

color: 'blue',

fontSize: '24px',

backgroundColor: 'lightgrey'

};

<h1 style={headingStyle}>Hello World</h1>

Or directly:

<h1 style={{ color: 'green', fontSize: '30px' }}>Styled Text</h1>

Inline styles are useful for dynamic styling or quick UI testing.

**11.ReactJS-HOL**

**1. Explain React Events**

Definition:  
React events are JavaScript events that are handled using React’s own event system. They are very similar to DOM events (like onclick, onchange, etc.) but follow a unified approach across all browsers using the Synthetic Event system.

Key Features:

* React events are wrapped in a cross-browser wrapper called SyntheticEvent.
* They are named using camelCase (onClick instead of onclick).
* You pass a function as the event handler (not a string).

Example:

<button onClick={handleClick}>Click Me</button>

Why use React events?  
They offer consistent behavior across all browsers and allow React to manage performance through event delegation.

**2. Explain About Event Handlers**

Definition:  
An event handler is a function that gets called when a specific event (like click, hover, change, etc.) occurs on a component.

How to define and use:

* Event handlers are typically defined as functions (arrow functions or regular functions).
* They are attached to elements using JSX attributes like onClick, onChange, etc.

Example:

function handleClick() {

alert("Button clicked!");

}

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

Best Practices:

* Name your handler clearly, like handleSubmit, handleChange, etc.
* Avoid binding this inside render(); use arrow functions or bind in constructor.

Event handlers help connect UI interactions to logic, enabling interactivity.

**3. Define Synthetic Event**

Definition:  
A SyntheticEvent is a React wrapper around the native browser event. It provides a consistent interface across all browsers and combines the behavior of different event systems into one.

Key Features:

* Automatically handled by React.
* Pooling is used to reuse event objects for performance (though this is deprecated in modern versions).
* Has the same interface as native events, including methods like preventDefault() and stopPropagation().

Example:

function handleSubmit(event) {

event.preventDefault(); // Prevents page reload

console.log("Form submitted");

}

Why use SyntheticEvent?

* Browser compatibility
* Optimized performance
* Unified event structure

**4. Identify React Event Naming Convention**

React uses a specific naming convention for events:

|  |  |  |
| --- | --- | --- |
| Feature | React | HTML |
| Naming | camelCase (onClick) | lowercase (onclick) |
| Handler Type | Function reference | String or function |
| Binding | Usually bound in code | Can be inline |

Examples:

* onClick instead of onclick
* onChange instead of onchange
* onSubmit, onMouseEnter, onKeyDown

Important Points:

* Always use camelCase.
* Do not put parentheses if you don’t want the function to execute immediately.

<button onClick={handleClick}>Click</button> (correct)

<button onClick={handleClick()}>Click</button> (incorrect)

**12.ReactJS-HOL**

**1. Explain about Conditional Rendering in React**

Definition:  
Conditional rendering in React means displaying different UI components or elements based on certain conditions (like state, props, or computed values). It allows React applications to make decisions about what to display at a given moment.

How it works:  
React uses standard JavaScript control flow (e.g., if, ? :, &&) inside JSX to implement conditional logic.

Common Techniques:

1. Using if statement:

if (isLoggedIn) {

return <Dashboard />;

} else {

return <Login />;

}

1. Using Ternary Operator:

{isLoggedIn ? <Dashboard /> : <Login />}

1. Using Logical AND (&&):

{isLoggedIn && <Welcome />}

1. Using Switch-Case inside a function:

function renderView(role) {

switch(role) {

case 'admin': return <AdminPanel />;

case 'user': return <UserPanel />;

default: return <GuestView />;

}

}

Why it’s important?  
It enables dynamic UI changes without reloading the page, improving user experience and interactivity.

**2. Define Element Variables**

Definition:  
Element variables are variables used to store JSX elements. These can be used to conditionally assign components or reuse UI elements, helping to simplify JSX and improve code readability.

Example:

let button;

if (isLoggedIn) {

button = <LogoutButton />;

} else {

button = <LoginButton />;

}

return (

<div>

<h1>Welcome</h1>

{button}

</div>

);

Advantages:

* Makes JSX cleaner and easier to read.
* Separates logic from the return statement.
* Useful when conditions are complex and JSX needs to be reused.

Use Case: Element variables are often used in place of inline conditions when rendering components conditionally within the render() or return block.

**3. Explain How to Prevent Components from Rendering**

Definition:  
Sometimes you may want to skip rendering a component entirely based on a condition (e.g., if data is not available or a user is not logged in). React allows you to conditionally return null or avoid rendering logic in such cases.

Methods to prevent rendering:

1. **Return null from a component:**

function WarningBanner(props) {

if (!props.showWarning) {

return null; // Nothing is rendered

}

return <div className="warning">Warning!</div>;

}

1. **Conditional rendering in JSX:**

{showComponent && <MyComponent />}

1. **Use early returns in functional components:**

function Profile({ user }) {

if (!user) return null;

return <div>Welcome, {user.name}</div>;

}

1. **Using shouldComponentUpdate() (in class components):**This lifecycle method allows preventing re-rendering if conditions aren't met.

shouldComponentUpdate(nextProps, nextState) {

return nextProps.visible; // Only re-render if visible is true

}

Why use it?  
To improve performance and avoid rendering unnecessary UI, which saves resources and makes the app more efficient.

**13.ReactJS-HOL**

**1. Explain various ways of conditional rendering**

Definition:  
Conditional rendering in React allows you to display different UI elements or components based on certain conditions like state, props, or computed values. It behaves like traditional control structures (if, switch, etc.) in JavaScript but is used inside JSX.

Ways to implement:

* If-else statement:  
  Use this when logic is more complex and requires multiple branches.
* Ternary operator (condition ? true : false):  
  Commonly used inside JSX when you want to quickly choose between two elements.
* Logical AND (&&) operator:  
  Renders content only if the condition is true. Useful for short conditional rendering.
* Immediately Invoked Function Expression (IIFE):  
  Use for inline conditional logic inside JSX when more control is needed.
* Switch-case:  
  Suitable when dealing with multiple possible conditions (like user roles).

*React re-renders only the parts of the UI that need to be updated, making conditional rendering very efficient.*

**2. Explain how to render multiple components**

Definition:  
In React, rendering multiple components means displaying more than one child component from a parent component.

Ways to render multiple components:

* Using JSX syntax: You can wrap multiple components inside a parent <div> or a React fragment (<> </>).
* React.Fragment or shorthand <> syntax:  
  Useful when you don’t want to add extra nodes to the DOM.

Example:

<>

<Header />

<Content />

<Footer />

</>

*Rendering multiple components promotes modular code and reuse, making apps more maintainable and scalable.*

**3. Define List Component**

Definition:  
A list component is a React component that renders a dynamic list of items, usually by iterating over an array using the map() function.

Use Cases:

* Rendering a list of users, products, tasks, or any collection.
* Dynamically showing results from an API.

Key Concepts:

* Use .map() to transform each item in an array into a JSX element.
* Assign a unique key to each list item for efficient updates.

*List components are fundamental when dealing with dynamic and data-driven UIs****.***

**4. Explain about keys in React applications**

Definition:  
A key is a special string attribute used by React to uniquely identify elements in a list and keep track of their state across re-renders.

Why are keys important?

* Improve performance: Helps React know which items changed, were added, or removed.
* Prevent unnecessary re-renders.
* Ensure stable component identity.

Best Practices:

* Use a unique and stable identifier (like an ID from a database).
* Avoid using array index as key unless no better option is available.

Example:

{items.map(item => <li key={item.id}>{item.name}</li>)}

*Without keys, React may re-render or mismanage component state in dynamic lists****.***

**5. Explain how to extract components with keys**

Definition:  
Extracting components with keys involves creating a reusable child component and rendering it in a list using a unique key.

Why extract?

* Separation of concerns
* Cleaner and reusable code
* Easier to maintain

Example:

function ListItem(props) {

return <li>{props.value}</li>;

}

function NumberList(props) {

return (

<ul>

{props.numbers.map(num => (

<ListItem key={num} value={num} />

))}

</ul>

);

}

Key Takeaways:

* key must be assigned at the time of list rendering, not inside the child component.
* Keys help React track changes and optimize re-renders**.**

**6. Explain React Map, map() function**

Definition:  
map() is a JavaScript function that creates a new array by applying a function to each item in the original array. In React, it is widely used to render elements or components dynamically.

Syntax:

array.map((item, index) => {

return <Component key={index} data={item} />;

});

Use in React:

* Convert an array of data into an array of JSX elements.
* Helps in rendering lists from APIs or data structures.

Benefits:

* Simplifies rendering of repetitive UI blocks.
* Works seamlessly with props and state.

Example:

const colors = ['Red', 'Green', 'Blue'];

const ColorList = () => (

<ul>

{colors.map((color, index) => (

<li key={index}>{color}</li>

))}

</ul>

);

*map() is essential in React for rendering lists efficiently and declaratively.*