1. What is React?

Answer:-
React is a **JavaScript library** developed by Facebook for building **user interfaces**, especially for single-page applications. It allows developers to build web apps that can update and render efficiently in response to data changes.

---

2. What are components in React?

Answer:
Components are **independent, reusable pieces of UI**. React supports:

- **Functional components** (using functions)
- **Class components** (using ES6 classes)

---

3. What is JSX?

Answer:
JSX stands for **JavaScript XML**. It's a syntax extension for JavaScript that looks similar to HTML and is used with React to describe what the UI should look like.

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

---

4. What is the virtual DOM?

Answer:
The **virtual DOM** is a lightweight JavaScript object that is a copy of the real DOM. React updates the virtual DOM first and compares it to the previous version (diffing). Then it updates the real DOM efficiently.

---

5. What are props and state?

Answer:

- **Props** (short for properties) are **read-only**, passed from parent to child.
- **State** is **mutable** and managed within the component itself using `useState` or class state.

---

6. What are hooks in React?

Answer:
Hooks are functions that let you **use state and other React features** in functional components. Common hooks include:

- `useState()`
- `useEffect()`
- `useContext()`
- `useRef()`

---

7. What is useEffect used for?

Answer:
`useEffect()` is a hook used to perform **side effects** (e.g., data fetching, subscriptions, manually updating the DOM) after a component renders.

```
useEffect(() => {
  fetchData();
}, []); // Empty array means it runs once after initial render
```

---

8. How does React handle forms?

Answer:
React uses **controlled components**, where form inputs are bound to component state.

```
const [value, setValue] = useState("");
<input value={value} onChange={(e) => setValue(e.target.value)} />
```

---

9. What is lifting state up?

Answer:
When multiple components need access to the same state, the state is moved up to their **nearest common ancestor** and passed down via props.

---

10. What is React Router?

Answer:
React Router is a standard library for routing in React. It enables **navigation between views** of various components using declarative routing.

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

11. What is Axios in React.js?

Axios is a **promise-based HTTP client** for the browser and Node.js used to **make HTTP requests** (like GET, POST, PUT, DELETE) to APIs. In **React.js**, it's commonly used to fetch data from a backend server or third-party APIs.

□ Why use Axios in React?

- Works in all modern browsers
- Automatically transforms JSON data
- Supports request and response **interceptors**
- Handles **errors** more cleanly than the native `fetch` API
- Allows setting **default headers**, timeouts, and base URLs

---

□ How to Install Axios

```
npm install axios
```

? Example: Using Axios in React

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const Users = () => {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/users')
```

```
      .then((response) => {
        setUsers(response.data);
      })
      .catch((error) => {
        console.error('Error fetching data:', error);
      });
  }, []);

  return (
    <ul>
      {users.map(user => <li key={user.id}>{user.name}</li>)}
    </ul>
  );
};

export default Users;
```

□ Axios Request Types

```
axios.get(url);
axios.post(url, data);
axios.put(url, data);
axios.delete(url);
```

12. What Are Hooks in React.js?

Hooks are **functions** that let you **"hook into" React features** (like state and lifecycle methods) **from functional components**.

Before hooks, only **class components** could manage state and side effects. Hooks allow functional components to do the same — in a cleaner and more reusable way.

□ Why Hooks?

Hooks were introduced in **React 16.8** to:

- Eliminate the need for class components
- Simplify logic reuse (no more higher-order components or render props)
- Organize code better by feature, not lifecycle methods

□ Commonly Used Hooks

Hook	Purpose
`useState`	Add **state** to a functional component
`useEffect`	Perform **side effects** (e.g. fetch data, timers)
`useContext`	Access **context** without `Consumer` wrapper
`useRef`	Persist a **mutable value** across renders (or access DOM elements)
`useMemo`	**Memoize** expensive calculations
`useCallback`	Memoize **functions** to prevent unnecessary re-renders
`useReducer`	Alternative to `useState`, useful for **complex state logic**

□ Example: `useState`

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0); // initial state is 0

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

□ Example: `useEffect`

```
import React, { useState, useEffect } from 'react';

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

  useEffect(() => {
    const timer = setInterval(() => setCount((c) => c + 1), 1000);
    return () => clearInterval(timer); // cleanup
  }, []);

  return <h1>Time: {count}s</h1>;
}
```

Rules of Hooks

1. **Only call hooks at the top level** (not inside loops, conditions, or nested functions)
2. **Only call hooks from React functions** (not regular JS functions)

13. What is ES6?

ES6 (ECMAScript 2015) is a major update to JavaScript that introduced many new features — including **arrow functions** and **classes** — which are widely used in modern JavaScript and React development.

□ 1. ES6 Functions (Arrow Functions)

□ Traditional Function:

```
function add(a, b) {
  return a + b;
}
```

□ ES6 Arrow Function:

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

□ Features of Arrow Functions:

- Shorter syntax
- Do **not have their own `this`** (they inherit it from their parent scope)
- Cannot be used as constructors (you can't use `new` with them)

□ Example in React:

```
const Greeting = () => {
  return <h1>Hello, World!</h1>;
};
```

□ 2. ES6 Classes

ES6 introduced **class-based syntax** to create objects and handle inheritance, replacing older `constructor function` patterns.

□ Basic Class Syntax:

```
class Person {
  constructor(name) {
    this.name = name;
  }

  greet() {
    return `Hello, my name is ${this.name}`;
  }
}

const user = new Person('Alice');
console.log(user.greet()); // Hello, my name is Alice
```

□ Example: React Class Component

Before hooks, state was only available in **class components**:

```
import React, { Component } from 'react';

class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  }

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>+</button>
      </div>
    );
  }
}
```

□ Summary: Arrow Functions vs Classes

Feature	Arrow Function	Class
Syntax	Short, concise	Longer, OOP-style
`this` binding	Inherits from parent scope	Bound to instance by default
React use	Functional components	Class components (pre-hooks era)
State support	Via `useState` hook	Via `this.state` and `setState`

14. Handle Forms in React

Handling forms in React is all about using **controlled components**, which means the form elements (like `<input>`, `<textarea>`, `<select>`) are controlled by **React state**.

□ Steps to Handle Forms in React

1. **Create state** to hold form values
2. **Bind input values** to that state
3. **Update state** on input change
4. **Handle form submission**

Here's a **very simple React form** example that adds **two numbers** entered by the user and displays the result.

□ Simple Addition Form in React

```
import React, { useState } from 'react';

function AddNumbers() {
  const [num1, setNum1] = useState('');
  const [num2, setNum2] = useState('');
  const [sum, setSum] = useState(null);

  const handleSubmit = (e) => {
    e.preventDefault();
    // Convert string inputs to numbers and add
    const result = parseFloat(num1) + parseFloat(num2);
    setSum(result);
  };

  return (
    <div>
      <h2>Add Two Numbers</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="number"
          placeholder="First number"
          value={num1}
          onChange={(e) => setNum1(e.target.value)}
        />
        <br /><br />
        <input
```

```
          type="number"
          placeholder="Second number"
          value={num2}
          onChange={(e) => setNum2(e.target.value)}
        />
        <br /><br />
        <button type="submit">Add</button>
      </form>

      {sum !== null && (
        <h3>Result: {sum}</h3>
      )}
    </div>
  );
}

export default AddNumbers;
```

□ How It Works:

- `useState` manages the values of `num1`, `num2`, and the `sum`.
- When the form is submitted, we **parse the input strings to numbers** using `parseFloat`.
- The result is displayed below the form.

15. What Are Props?

Props (short for “properties”) are how **data is passed** from a **parent component to a child** component in React.
Props are **read-only** in the child.

Example: Passing Data Using Props

`App.js` (Parent Component)

```
import React from 'react';
import Greeting from './Greeting'; // Importing the child component

function App() {
  const userName = "John Doe";

  return (
    <div>
      <h1>Welcome to React Props Example</h1>
      <Greeting name={userName} />  {/* Passing "name" as a prop */}
    </div>
  );
}

export default App;
```

`Greeting.js` (Child Component)

```
import React from 'react';

function Greeting(props) {
  return (
    <h2>Hello, {props.name}!</h2>
  );
}

export default Greeting;
```

Output on the Page

```
Welcome to React Props Example
Hello, John Doe!
```

□ Props in One Line

□ Props let a **parent send data** → □ Child uses `props` to **display or act on that data**.

□ Extra: Using Destructuring for Props

In the child component, instead of writing `props.name`, you can destructure:

```
function Greeting({ name }) {
  return <h2>Hello, {name}!</h2>;
}
```

Let's look at **3 examples** of passing **different types of props** in React:

□ 1. Passing an Object as Props

□ `App.js` (Parent Component)

```
import React from 'react';
import UserInfo from './UserInfo';

function App() {
  const user = {
    name: "Alice",
    age: 28,
    email: "alice@example.com"
  };

  return (
    <div>
      <UserInfo user={user} />
    </div>
  );
}
```

```
export default App;
```

📄 `UserInfo.js` (Child Component)

```
import React from 'react';

function UserInfo({ user }) {
  return (
    <div>
      <h2>User Info</h2>
      <p>Name: {user.name}</p>
      <p>Age: {user.age}</p>
      <p>Email: {user.email}</p>
    </div>
  );
}

export default UserInfo;
```

🔹 2. Passing an Array as Props

📄 `App.js`

```
import React from 'react';
import ShoppingList from './ShoppingList';

function App() {
  const items = ["Apples", "Bananas", "Oranges"];

  return (
    <div>
      <ShoppingList items={items} />
    </div>
  );
}

export default App;
```

📄 `ShoppingList.js`

```
import React from 'react';

function ShoppingList({ items }) {
  return (
    <div>
      <h2>Shopping List</h2>
      <ul>
        {items.map((item, index) => (
```

```
          <li key={index}>{item}</li>  // Always add a unique key in lists
        ))}
      </ul>
    </div>
  );
}

export default ShoppingList;
```

□ 3. Passing a Function as a Prop (Callback)

□ App.js

```
import React from 'react';
import Button from './Button';

function App() {
  const handleClick = () => {
    alert('Button clicked from child component!');
  };

  return (
    <div>
      <Button onClick={handleClick} />
    </div>
  );
}

export default App;
```

□ Button.js

```
import React from 'react';

function Button({ onClick }) {
  return (
    <button onClick={onClick}>Click Me</button>
  );
}

export default Button;
```

□ Summary

Type	How to Use
Object	Pass as a single prop and access fields
Array	Map through it in the child
Function	Pass it as a prop and call it in child