# React

**Day 1 – Introduction to React**

**Q1. What is React?**
A: React is a JavaScript library for building user interfaces using a component-based architecture.

**Q2. Why should we use React?**
A: It offers fast rendering via Virtual DOM, reusable components, and is great for building Single Page Applications (SPAs).

**Q3. How do you create a React app?**
A: Use the command: **npx create-react-app my-app**

4. **Key Features:**

- Component-based
- Virtual DOM
- Unidirectional data flow

Read Documentions and Refer GeeksForGeeks like websites you tube videos for Introduction To React

```
function Welcome() {
  return <h1>Hello, React Beginner!</h1>;
}


export default function App() {
  return (
    <div>
      <Welcome />
    </div>
  );
}
```

**Day 2 – JSX and Components**

**Q1. What is JSX?**
A: JSX stands for JavaScript XML. It allows writing HTML in React.

**Q2. How do you define a functional component?**

Use Function keyword to define functional components

```
function Greet() {
  return <h1>Hello</h1>;
}
```

**Q3. What is the difference between functional and class components?**
A: Functional components are functions, class components use class syntax and lifecycle methods.

| Functional Component | Class Component |
|---|---|
| Uses function | Uses class |
| Uses hooks | Uses lifecycle methods |
| Shorter code | More boilerplate |

```
function Greeting() {
  const name = "Ameena";
  return <h2>Hello, {name}! Welcome to JSX.</h2>;
}

export default function App() {
  return (
    <div>
      <Greeting />
    </div>
  );
}
```

**Day 3 – Props in React**

**Q1. What are props?**
A: Props are short for “properties”. They allow data to be passed from parent to child components.

**Q2. How to pass props?**

```
<Greet name="Ameena" />
```

**Q3. How to receive props?**

```
function Greet(props) {
  return <h1>Hello {props.name}</h1>;
}
function GreetUser(props) {
  return <h2>Hello, {props.name}!</h2>;
}


export default function App() {
  return (
    <div>
      <GreetUser name="Ameena" />
      <GreetUser name="Zara" />
    </div>
  );
}
```

**Q4. Props are immutable** – you cannot change their values inside child components.

**Day 4 – State in React**

**Q1. What is state?**
A: State is a built-in object that stores property values that belong to a component**.**

**Q1. What is useState?**
A: useState is a React Hook to manage state in functional components.

**Q2. useState Hook Syntax:**

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

**Q3. Updating State:**

```
setCount(count + 1);
```

**Q4. Difference between state and props:**

Props: Read-only, passed from parent

State: Mutable, maintained by component itself

```
import React, { useEffect, useState } from 'react';
function Timer() {
  const [time, setTime] = useState(new Date().toLocaleTimeString());
  useEffect(() => {
    const timer = setInterval(() => {
      setTime(new Date().toLocaleTimeString());
    }, 1000);
    return () => clearInterval(timer); // cleanup
  }, []);
  return (
    <div>
      <h2>Current Time: {time}</h2>
    </div>
  );
}
export default Timer;
```

**Day 5: useEffect and Fetch**

**Q1. What is useEffect used for?**
A: It handles side-effects like data fetching, updating DOM, or setting timers.

**Q2. Example – Fetch data using useEffect:**

```
useEffect(() => {
  fetch('https://api.example.com')
    .then(res => res.json())
    .then(data => setData(data));
}, []);
```

| **useState** | **useEffect** |
| --- | --- |
| Stores data/state | Runs code after rendering |
| Triggers re-render | Triggers side effects |

**4. Dependency Array**

- [] → run only once (like componentDidMount)
- [variable] → run when that variable changes
- No array → run on **every render**

```
useEffect(() => {
  console.log('Runs once');
}, []);
```

Example: Fetch users from an API

```
import React, { useEffect, useState } from 'react';
function UserList() {
  const [users, setUsers] = useState([]);
  useEffect(() => {
    // API call
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => response.json())
      .then(data => setUsers(data));
  }, []);

  return (
    <div>
      <h2>Users</h2>
      <ul>
        {users.map(user => (
          <li key={user.id}>{user.name} ({user.email})</li>
        ))}
      </ul>
    </div>
  );
}

export default UserList;
```

2. Create a timer using useEffect + setInterval Display current time or tick count

```
import React, { useEffect, useState } from 'react';
function UserList() {
  const [users, setUsers] = useState([]);
  const [time, setTime] = useState(new Date().toLocaleTimeString());
  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(res => res.json())
      .then(data => setUsers(data))
      .catch(err => console.error('Error fetching users:', err));
  }, []);
  useEffect(() => {
    const timer = setInterval(() => {
      setTime(new Date().toLocaleTimeString());
    }, 1000);
    // Cleanup interval when component unmounts
    return () => clearInterval(timer);
  }, []);
  return (
    <div style={{ padding: '20px', fontFamily: 'sans-serif' }}>
      <h2>Current Time: {time}</h2>
      <h3>User Names:</h3>
      <ul>
        {users.map(user => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}
export default UserList;
```

**Day 6: Forms, Events, and Controlled Components**

**1. Introduction to Forms in React**

**Q1. How are forms handled in React compared to plain HTML?**
**A:** In plain HTML, form elements maintain their own state. In React, form elements are controlled by the component's state using useState, making them **controlled components**.

**Q2. What are controlled components in React?**
**A:** A controlled component is a form element (like <input>, <textarea>, or <select>) whose value is controlled by React state, using value and onChange

**2. Controlled Components**

**Q3. What is the purpose of onChange in a controlled component?**
**A:** onChange updates the component's state whenever the input changes, ensuring the displayed value is always in sync with the state.

**Q4. Give a simple example of a controlled component.**

```
const [name, setName] = useState('');

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

**Example:**

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

function ControlledForm() {
  const [name, setName] = useState('');
  const handleChange = (e) => {
    setName(e.target.value);};
  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Submitted Name: ${name}`);
  };
  return (
    <form onSubmit={handleSubmit}>
      <input type="text" value={name} onChange={handleChange} />
      <button type="submit">Submit</button>
    </form>
  );}


export default ControlledForm;
```

**3. Handling Multiple Inputs**

**Q5. How do you handle multiple inputs in one form in React?**
**A:** Use a single state object (like formData) and update it dynamically using the name attribute of each input:

**Q6. Why use [e.target.name] in the handleChange function?**
**A:** It dynamically accesses the key in the state object based on the input field's name attribute.

```
function MultiInputForm() {
  const [formData, setFormData] = useState({ username: '', email: '' });
  const handleChange = (e) => {
    setFormData({
      ...formData,
      [e.target.name]: e.target.value,
    });
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    console.log(formData);
  };
  return (
    <form onSubmit={handleSubmit}>
      <input name="username" value={formData.username} onChange={handleChange} />
      <input name="email" value={formData.email} onChange={handleChange} />
      <button type="submit">Submit</button>
    </form>
  );
}
```

**4. Event Handling in React**

**Q7. What is SyntheticEvent in React?**
**A:** SyntheticEvent is a cross-browser wrapper around the browser's native event, providing consistent behavior across different browsers.

**Q8. How do you prevent default form submission behavior in React?**
**A:** By calling e.preventDefault() inside the form's onSubmit handler.

```
function ClickButton() {
  const handleClick = () => {
    alert('Button Clicked!');
  };

  return <button onClick={handleClick}>Click Me</button>;
}
```

Q10. How can you reset form fields after submission in React?

```
setFormData({ name: '', age: '' });
```

**Practice: Build a Simple Form Component**

- **Task:**

1. Create a form with:
   - Name input
   - Age input
   - A submit button
2. On submit, show an alert with entered values.

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

function SimpleForm() {
  const [formData, setFormData] = useState({
    name: '',
    age: ''
  });
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prevData) => ({
      ...prevData,
      [name]: value
    }));
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Name: ${formData.name}\nAge: ${formData.age}`);
  };
  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input
          type="text"
          name="name"
          value={formData.name}
```

```
        onChange={handleChange}
      />
    </label>
    <br />
    <label>
      Age:
      <input
        type="number"
        name="age"
        value={formData.age}
        onChange={handleChange}
      />
    </label>
    <br />
    <button type="submit">Submit</button>
  </form>
 );
}

export default SimpleForm;
```

**Day 7: Handling Events & Conditional Rendering in React**

**1. Handling Events in React**

**Q1. How is event handling in React different from HTML?**
**A:**React uses **camelCase** (onClick) instead of lowercase (onclick).

React functions are passed **directly**, not as strings.

**Q2. Give an example of handling a button click in React.**

```
import React from 'react';

function EventExample() {
  const handleClick = () => {
    alert('Button Clicked!');
  };
  return <button onClick={handleClick}>Click Me</button>;
}

export default EventExample;
```

**2. Passing Arguments to Event Handlers**

**Q3. How do you pass parameters to an event handler in React?**
**A:** Use an arrow function:

```
<button onClick={() => greetUser('Ameena')}>Greet</button>

function greetUser(name) {
  alert(`Hello, ${name}`);
}
```

**Q4. Why not call greetUser('Ameena') directly in onClick?**
**A:** Because it will **execute immediately** when the component renders, not when the button is clicked.

**3. Conditional Rendering**

**Q5. What are the ways to conditionally render in React?**
**A:**

- if...else statements
- Ternary operator condition ? A : B
- Logical AND condition && A

**Q6. Example: How to display "Welcome" only if isLoggedIn is true?**

```
{isLoggedIn && <h1>Welcome</h1>}
```

**Q7. What does the ternary operator do in rendering?**
**A:** It chooses between two elements based on a condition:

{isLoggedIn ? 'Logout' : 'Login'}

**Q1. How do you conditionally render components in React?**
A: Using if statements or ternary operator

**4. Toggle Button Logic**

**Q8. How do you toggle login state using a button in React?**
**A:** Use useState and flip the boolean value with setIsLoggedIn(!isLoggedIn).

**Practice Task**

Build a simple greeting component:

- Input box: Enter name
- Button: "Greet"
- Show: “Hello, [Name]!” below when button is clicked

Want help with the code?

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

function GreetingComponent() {
  const [name, setName] = useState('');
  const [greetedName, setGreetedName] = useState('');
  const handleChange = (e) => {
    setName(e.target.value); };
  const handleGreet = () => {
    setGreetedName(name); };
  return (
    <div>
      <input
        type="text"
        placeholder="Enter your name"
        value={name}
        onChange={handleChange}; />
      <button onClick={handleGreet}>Greet</button>
      {greetedName && <p>Hello, {greetedName}!</p>}
    </div>);}


export default GreetingComponent;
```

**Day 8: Lists and Keys in React**

3. **What are Lists in React?**

A:Lists in React are arrays of data used to render repeating UI elements like <li>, , or components using .map().

4. **How do you render a list in React?**

A:Use the .map() method to loop through an array and return JSX:

```
import React from 'react';
function NameList() {
  const names = ['Ameena', 'Zara'];
  const listItems = names.map((name, index) => <li key={index}>{name}</li>);

  return (
    <div>
      <h2>List of Names</h2>
      <ul>
        {listItems}
      </ul>
    </div>
  );
}
```

**export default NameList;**

**3. What are Keys in React Lists?**

**A:**Keys are special props used to uniquely identify list elements. They help React optimize rendering by tracking which items change.

✅ **4. Why should you avoid using array indexes as keys?**

A:Because if the list order changes or items are removed, index-based keys can cause UI bugs or incorrect element reuse.

✅ **5. What's the best way to assign keys in a list?**

**A:**Use stable, unique identifiers from your data, like user.id, instead of index or Math.random().

**Assignment for Practice:**

Create a list of users with name, email, and role.

Display them using .map() in a React component.

Use user.id as key.

```
import React from 'react';

const users = [
  { id: 1, name: 'Ameena', email: 'ameena@example.com', role: 'Admin' },
  { id: 2, name: 'Zara', email: 'zara@example.com', role: 'Editor' },
  { id: 3, name: 'John', email: 'john@example.com', role: 'Viewer' },
];


function UserList() {
  return (
    <div>
      <h2>User List</h2>
      <ul>
        {users.map((user) => (
          <li key={user.id}>
            <strong>{user.name}</strong> – {user.email} – <em>{user.role}</em>
          </li>
        ))}
      </ul>
    </div>
  );
}


export default UserList;
```