React Documentation — useState Hook

1. Introduction

```
useState is a React Hook that allows functional components to maintain
and manage local state.
```

State represents dynamic data that changes over time and triggers UI updates when modified.

Before Hooks, state management was only available in class components.

2. Importing useState

```
import { useState } from "react";
```

3. Syntax

```
const [state, setState] = useState(initialValue);
```

4. Basic Usage

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

return (

```
    <button onClick={() => setCount(count + 1)}>
      {count}
```

</button>

);

```
}
```

5. State Update Rules

State is immutable. Never modify state directly.

Incorrect: count = count + 1;

Correct: setCount(count + 1);

```
Calling the setter function causes React to update state and re-render
the component.
```

6. Functional Updates

Use when new state depends on previous state:

```
  setCount(prev => prev + 1);
```

Useful for async logic and avoiding stale values.

7. Multiple State Variables

```
  const [name, setName] = useState("Harshal");
  const [age, setAge] = useState(22);
```

8. Object State

```
  const [user, setUser] = useState({
name: "Harshal",
age: 22,
city: "Pune"
  });
```

Incorrect:

```
  setUser({ age: 23 });
```

Correct:

```
  setUser({
...user,
age: 23
  });
```

9. Nested Object Updates

```
  setUser(prev => ({
...prev,
    address: {
...prev.address,
city: "Mumbai"
    }
  }));
```

10. Array State

```
  const [items, setItems] = useState([]);
```

Add:

```
  setItems(prev => [...prev, "New Item"]);
```

Remove:

```
  setItems(prev => prev.filter(item => item !== "A"));
```

Update:

```
  setItems(prev =>
    prev.map(item =>
item === "A" ? "Updated" : item
)
);
```

11. Array of Objects

```
  const [users, setUsers] = useState([
    { id: 1, name: "Harshal" },
    { id: 2, name: "Rahul" }
]);
```

```
  setUsers(prev =>
    prev.map(user =>
user.id === 1
        ? { ...user, name: "New Name" }
: user
)
);
```

12. Controlled Inputs

```
  const [input, setInput] = useState("");
```

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

13. Immutability Concept

React compares references to detect changes.

Incorrect:

array.push(newItem);

setArray(array);

Correct:

setArray([...array, newItem]);

14. Mental Model

React state behaves like a snapshot.

Do not mutate existing state.

Always create new object or array.

Use spread operator frequently.

15. Common Mistakes

- Direct mutation of state
- Forgetting spread operator
```
  - Using old state instead of functional update
```
- Expecting immediate update after setState

16. Best Practices

- Keep state minimal
- Prefer multiple states instead of deeply nested objects
```
  - Use functional updates when possible
```
- Avoid unnecessary re-renders