Ternary and Short-Circuit Rendering

In React, **ternary rendering** and **short circuit rendering** are two common ways to conditionally render elements based on the component's state or props. Here's a detailed explanation of both — without code:

Ternary Rendering

What it is:

Ternary rendering uses the **ternary operator** (`condition ? expr1 : expr2`) to decide which JSX to render.

How it works:

- It evaluates a condition.
- If the condition is true, one block of JSX is rendered.
- If the condition is false, another block is rendered.

Why use it:

- It's useful when you want to render **one of two** distinct elements or UI states.
- It makes logic explicit — "if this, then show this; else, show that."

Example scenarios:

- Show a "Login" button if the user is logged out, and a "Logout" button if the user is logged in.
- Show different layouts for mobile vs. desktop views.

Pros:

- Clear logic for binary conditions.
- Good for readability when the condition is not too complex.

Cons:

- Can become hard to read if overused or nested deeply.
- Not ideal for multi-condition rendering unless handled carefully.

Short Circuit Rendering

What it is:

Short circuit rendering uses logical operators (`&&` and `||`) to conditionally render JSX.

Commonly used with `&&`:

- Only renders a component if the condition is true.
- If the condition is false, nothing is rendered (returns `false` or `null`).

How it works:

- With `condition && JSX`, React evaluates the condition.
    - If it's true, the JSX is rendered.
    - If it's false, React ignores the JSX (nothing is shown).

Why use it:

- It's very concise for rendering something only when a condition is true.
- Often used to avoid writing full if-else logic when the "else" case is empty.

Pros:

- Very concise.
- Clean for simple conditional rendering (especially for one-sided conditions).

Cons:

- Doesn’t handle “else” cases (unlike ternary).
- Can be confusing if the left-hand side expression doesn't return a pure boolean (e.g., `0` or empty strings may unintentionally suppress rendering).

Program:

```
import './App.css'

import { useState } from "react";

function App() {

  const [isLoggedIn, setIsLoggedIn] = useState(false);

  const [username, setUsername] = useState('John Doe');

  return (

    <div className='App'>

      <h1>Welcome to the App!</h1>

      {/* 🔁 Toggle login status */}

      <button onClick={() => setIsLoggedIn(!isLoggedIn)}>

        {isLoggedIn ? 'Logout' : 'Login'}

      </button>

      <hr />

      {/* ✅ Ternary Rendering: Show message based on login */}

      <p>

        {isLoggedIn ? `Hello, ${username}!` : 'Please log in to
continue.'}

      </p>
```

```
      {/* ✅ Short-Circuit Rendering: Only shows when logged in */}
      {isLoggedIn && (
        <div className='dashboard'>
          <h3>Dashboard</h3>
          <p>Here's your private dashboard info.</p>
        </div>
      )}
    </div>
  );
}
export default App;
```

Rendering Lists:

Rendering lists in React refers to the process of **dynamically generating multiple elements** (like `<div>`, `<li>`, or custom components) based on the items in an **array or a collection of data**.

Instead of writing out each element manually, React lets you **loop over a list** of data and render each item efficiently using JSX.

Why Do We Need It?

In most real-world apps, data comes in the form of arrays — lists of:

- Products
- Users
- Messages
- Posts
- Notifications
- etc.

Hardcoding each item isn't scalable. Instead, we use list rendering to:

- **Automatically create UI elements for each data item**
- **Keep UI in sync** with dynamic or changing data

How It Works Conceptually

React allows you to take a **data array**, and for each item:

1. **Iterate** over it (typically with `.map()` in JavaScript).
2. **Return JSX** for each item.
3. React compiles that into a virtual DOM list, and then renders it on the page.

```
import React from 'react';



function App() {

  const fruits = ['Apple', 'Banana', 'Cherry', 'Mango'];



  return (

    <div style={{ padding: '20px', fontFamily: 'Arial' }}>

      <h2>Fruit List (with Key)</h2>



      {/* ✅ Proper usage with key */}

      <ul>

        {fruits.map((fruit, index) => (

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

        ))}

      </ul>
```

```
      <hr />


      <h2>Fruit List (without Key)</h2>


      {/* ⚠️ Without key — React will warn in console */}
      <ul>
        {fruits.map((fruit) => (
          <li>{fruit}</li>
        ))}
      </ul>
    </div>
  );
}
export default App;
```

Dynamic List

A **dynamic list** in React refers to a list of UI elements that is **generated and updated based on changing data** — not hardcoded.

Key Characteristics of a Dynamic List

1. **Data-driven**:
   The list is built from an array of objects or values stored in **state**, **props**, or received from an API.
2. **Automatically rendered**:
   React uses `.map()` to render UI for each item in the list. When the data

changes, the UI re-renders automatically.

3. **Interactive**:
   You can add features like:

   - Sorting
   - Filtering
   - Pagination
   - Search
   - Live updates (e.g., chat messages or notifications)

4. **Reactive**:
   React's core feature is reactivity. If the data changes (e.g., via `setState`), the list on the screen updates without manual DOM manipulation.

```jsx
import StudentList from './StudentList.jsx';

function App() {

  const students = [
    { id: 1, name: "Alice", score: 98, rank: 1 },
    { id: 2, name: "Bob", score: 91, rank: 2 },
    { id: 3, name: "Charlie", score: 85, rank: 3 },
    { id: 4, name: "David", score: 80, rank: 4 },
    { id: 5, name: "Eva", score: 76, rank: 5 },
    { id: 6, name: "Frank", score: 70, rank: 6 },
    { id: 7, name: "Grace", score: 65, rank: 7 }
  ];

  // 🔍 Filter top 5 ranked students
  const topFiveStudents = students
    .filter(student => student.rank <= 5)
    .sort((a, b) => a.rank - b.rank); // Optional sort by rank

  return (
    <>
      <h1 className="title">🏆 Top 5 Student Rankings</h1>
      <StudentList students={topFiveStudents} />
    </>
  );
```

```
}

export default App;
```

```
import './index.css'

function StudentList({ students = [] }) {
    // Sort students by rank (ascending)
    const sortedStudents = [...students].sort((a, b) => a.rank - b.rank);

    // Get badge based on rank
    const getBadge = (rank) => {
      if (rank === 1) return "🥇 Gold";
      if (rank === 2) return "🥈 Silver";
      if (rank === 3) return "🥉 Bronze";
      return "🎖️ Participant";
    };

    const listItems = sortedStudents.map((student) => (
      <li key={student.id} className="student-item">
        <span className="rank">#{student.rank}</span>
        <span className="name">{student.name}</span>
        <span className="score">Score: {student.score}</span>
        <span className="badge">{getBadge(student.rank)}</span>
      </li>
    ));

    return <ol className="student-list">{listItems}</ol>;
  }

  export default StudentList;
```

```css
.title {
  text-align: center;
  font-size: 2.5em;
  margin-bottom: 20px;
}
.student-list {
  list-style: none;
  padding: 0;
  max-width: 600px;
  margin: auto;
}
.student-item {
  font-size: 1.5em;
  display: flex;
  justify-content: space-between;
  padding: 12px 20px;
  margin: 10px 0;
  border: 2px solid #ccc;
  border-radius: 8px;
  background-color: #f8f8f8;
}
.rank {
  font-weight: bold;
  width: 60px;
}
.name {
  flex-grow: 1;
  text-align: left;
}

.score {
  width: 120px;
  text-align: right;
}

.badge {
  font-weight: bold;
  margin-left: 10px;
}
```