

# REACT NOTES

---

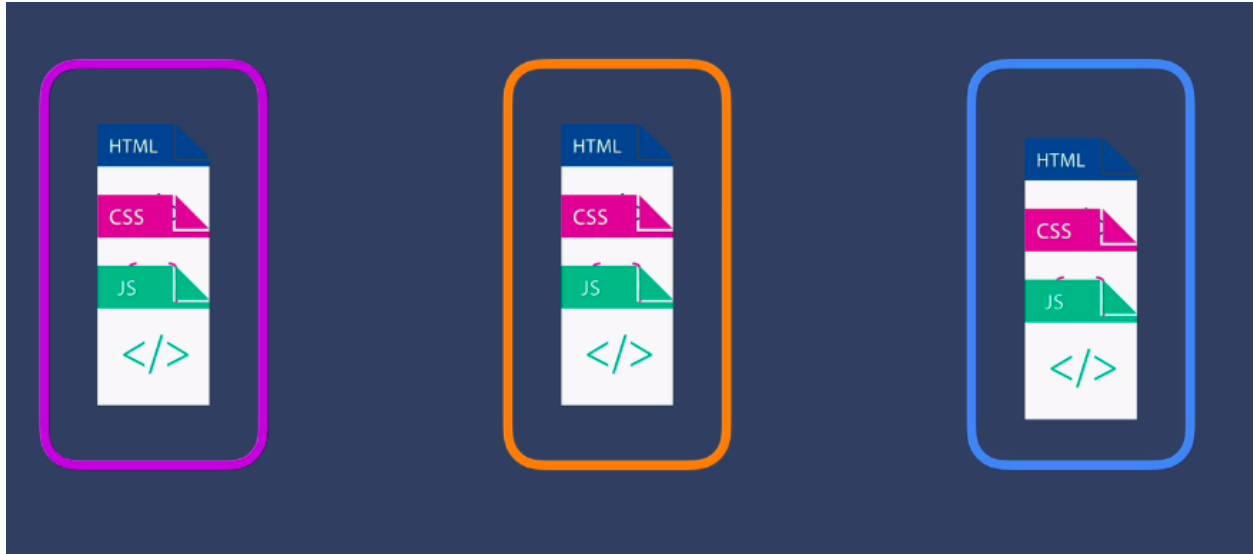
## **Table of Contents :**

- 1. Introduction**
- 2. How does React work**
  - a. Create React App
  - b. Run the React Application
- 3. Introduction to JSX**
  - a. React Render HTML
  - b. React JSX and Babel
- 4. Javascript Expressions in JSX & ES6 Template Literals**
- 5. JSX Attributes & Styling React Elements**
  - a. Inline Styling for React Components
- 6. React Components**
- 7. Javascript ES6 - Import, Export and Modules**
- 8. React Props**
  - a. React DEV TOOLS
  - b. Mapping Data to Components
- 9. Javascript ES6 Map/Filter/Reduce**
- 10. Javascript ES6 Arrow Functions**
- 11. React Conditional Rendering with the Ternary operator & AND operator**
- 12. React Hooks**
  - a. useState
  - b. useEffect
- 13. JavaScript ES6 Object & Array Destructuring**
- 14. Event Handling in React**
- 15. React Forms**
- 16. Class Components vs Functional Components**
- 17. Changing Complex State**
- 18. JavaScript ES6 Spread Operator**
- 19. React Context API and useContext hook**
  - a. Global State and Lifting up the State
  - b. PropDrilling
  - c. Context API
  - d. useContext Hook
- 20. React Router**
- 21. React API Integration - Fetch API**
- 22. Local Storage**

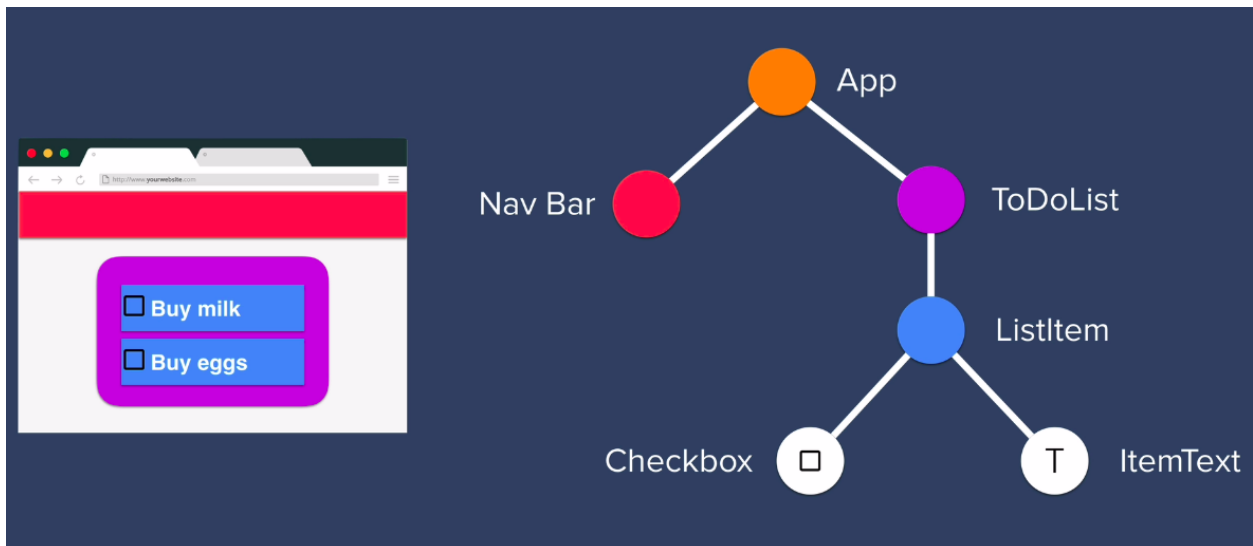
## Introduction :

- **React** is a **JavaScript library** for building user interfaces. (Frontend Framework)
- React is used to build **single-page applications** (A single-page application (SPA) is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages).
- React allows us to create **reusable UI components**.

The below diagram shows different **components** (combination of HTML, CSS and JavaScript).

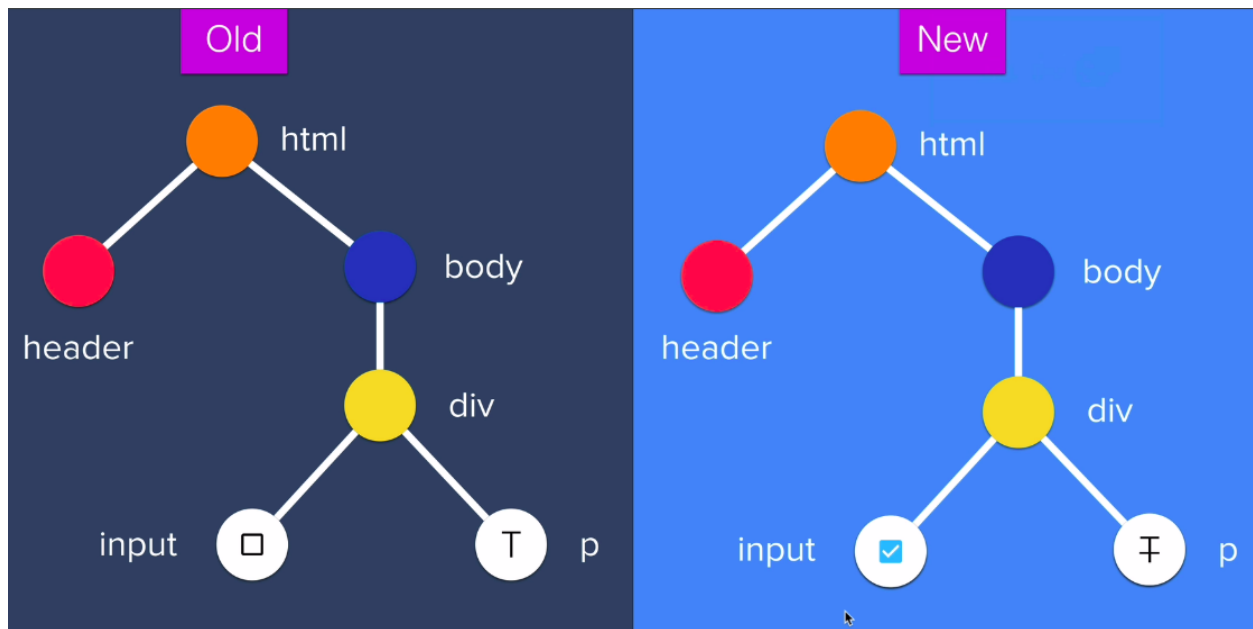


User Interface structure to a Component tree.



React is able to re-render the changes effectively by comparing old and new DOM trees. It is also called **diffing** (The process of checking the difference between the new VDOM tree and the old VDOM tree is called "diffing"). VDOM - Version DOM

The below diagram compares the component tree of old and new DOM tree. The new version will detect the changed components like the input and the text.



## How does React work :

**React creates a VIRTUAL DOM in memory.**

→ Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.

**React only changes what needs to be changed!**

→ React finds out what changes have been made, and changes only what needs to be changed.

## Create React App:

- The **create-react-app** tool is an officially supported way to create React applications.
- Node.js is required to use create-react-app.
- Run the below command to create a React application named my-react-app:

```
npx create-react-app my-react-app
```

## Run the React Application:

- Run the below commands

```
cd my-react-app
npm start
```

## Introduction to JSX :

### Dependencies/Modules installed :

- react
- react-dom

```
import React from "react";
import ReactDOM from "react-dom";

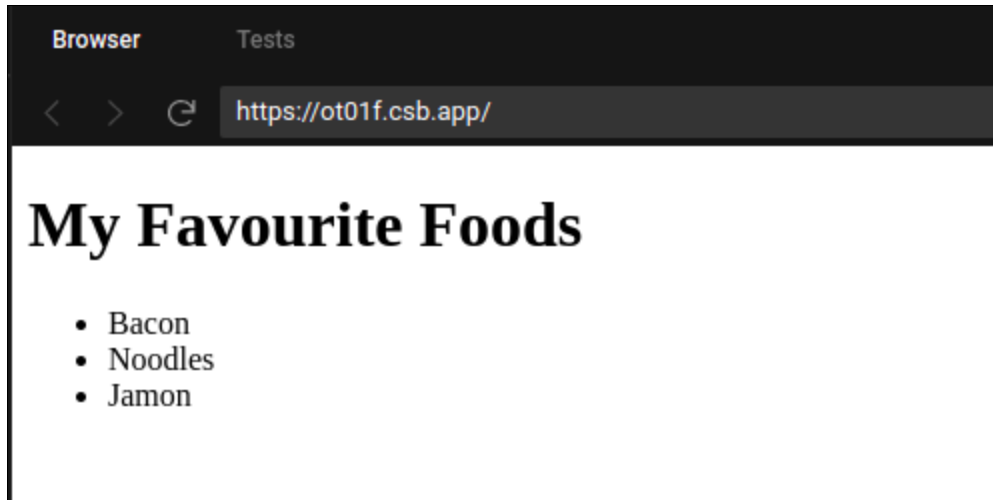
// ReactDOM.render("WHAT TO SHOW", "WHERE TO SHOW IT");
ReactDOM.render(
  <div>
    <h1>Hello World!</h1>
    <p>This is a paragraph.</p>
  </div>,
  document.getElementById("root")
);
```

Code link : <https://codesandbox.io/s/introduction-to-jsx-oofqn?fontsize=14&file=/src/index.js>

JSX Code Practice : <https://codesandbox.io/s/jsx-code-challenge-completed-ot01f?fontsize=14>

```
import React from "react";
import ReactDOM from "react-dom";

ReactDOM.render(
  <div>
    <h1>My Favourite Foods</h1>
    <ul>
      <li>Bacon</li>
      <li>Noodles</li>
      <li>Jamon</li>
    </ul>
  </div>,
  document.getElementById("root")
);
```



## React Render HTML :

- React renders HTML to the web page by using a function called ReactDOM.render().
- The ReactDOM.render() function takes two arguments, HTML code and an HTML element.
- The purpose of the function is to display the specified HTML code inside the specified HTML element.
- But render where?
- There is another folder in the root directory of your React project, named "public". In this folder, there is an index.html file.
- You'll notice a single <div> in the body of this file. This is where our React application will be rendered.

## React JSX :

- JSX stands for JavaScript XML.
- JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods.
- JSX converts HTML tags into react elements.

## Babel :

Converts code to Javascript code:

<pre>1 ReactDOM.render(&lt;h1&gt;Hello World!&lt;/h1&gt;,   document.getElementById("root"));</pre>	<pre>1 "use strict"; 2 3 ReactDOM.render(React.createElement("h1", null, "Hello World!"),   document.getElementById("root"));</pre>
---	---

# Babel is a JavaScript compiler.

Use next generation JavaScript, today.

Babel 7 is out! Please read our announcement and upgrade guide for more information.

Put in next-gen JavaScript

```
var obj = {  
  shorthand,  
  method() {  
    return "👉";  
  }  
};
```

Get browser-compatible JavaScript out

```
var obj = {  
  shorthand: shorthand,  
  method: function method() {  
    return "👉";  
  }  
};
```

## JavaScript Expressions in JSX & ES6 Template Literals:

JSX allows us to write HTML inside Javascript. Now, we can also add Javascript code inside HTML.



But we cannot insert statements inside the HTML element (e.g if conditions), we can only insert expressions which can be evaluated.

```
import React from "react";
import ReactDOM from "react-dom";

const fName = "Angela";
const lName = "Yu";
const num = 7;

ReactDOM.render(
  <div>
    <h1>Hello {fName + " " + lName}!</h1>
    <p>Your lucky number is {num}</p>
  </div>,
  document.getElementById("root")
);
```

**Code link :**

<https://codesandbox.io/s/javascript-expressions-in-jsx-fmhnrm?fontsize=14&file=/src/index.js>

## **JSX Attributes & Styling React Elements:**

**index.js :** (Notice the camelCase name Styling)

```
import React from "react";
import ReactDOM from "react-dom";

ReactDOM.render(
  <h1 className="heading">My Favourite Foods</h1>,
  document.getElementById("root")
);
```

**index.html :** (Changed the script type from Javascript to JSX)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JSX</title>
    <link rel="stylesheet" href="styles.css" />
  </head>

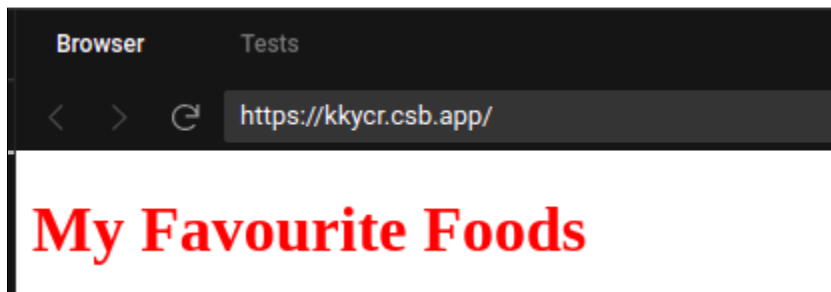
  <body>
```

```
<div id="root"></div>
<script src="../../src/index.js" type="text/Javascript"></script>
</body>
</html>
```

#### styles.css

```
.heading {
  color: red;
}
```

Output:



Code link :

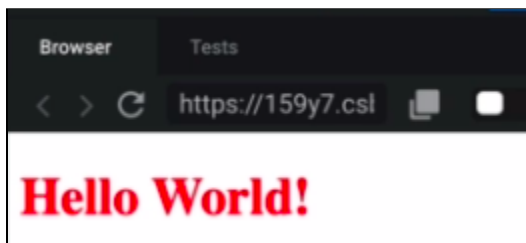
<https://codesandbox.io/s/jsx-attributes-and-styling-kkyocr?fontsize=14&file=/src/index.js>

### Inline Styling for React Components:

First brace is for a Javascript object and second is for inserting JS code inside HTML.

```
import React from "react";
import ReactDOM from "react-dom";

ReactDOM.render(
  <h1 style={{color: "red"}}>Hello World!</h1>,
  document.getElementById("root")
);
```



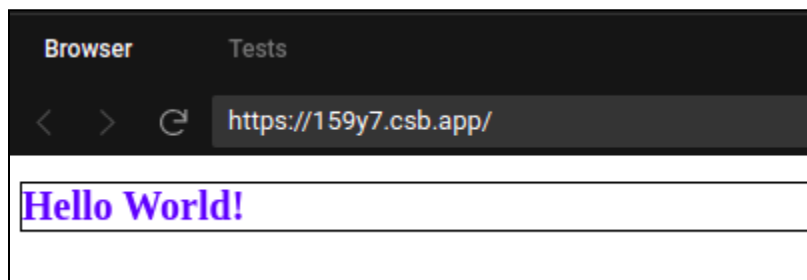


```
import React from "react";
import ReactDOM from "react-dom";

const customStyle = {
  color: "red",
  fontSize: "20px",
  border: "1px solid black"
};

customStyle.color = "blue";

ReactDOM.render(
  <h1 style={customStyle}>Hello World!</h1>,
  document.getElementById("root")
);
```



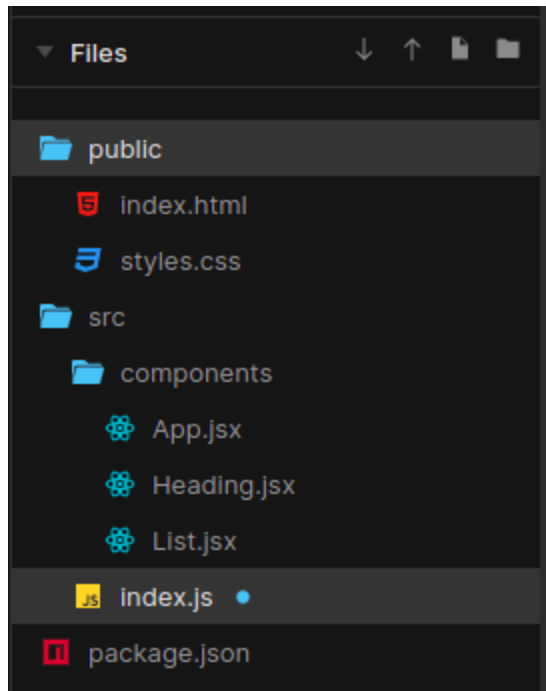
**Code link :** <https://codesandbox.io/s/inline-styling-in-jsx-159y7?fontsize=14&file=/src/index.js>

**Practice :** <https://codesandbox.io/s/react-styling-practice-xpn8u?fontsize=14&file=/src/index.js>

## **React Components :**

- Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.
- Components come in two types, Class components and Function components, but we will concentrate on Function components.
- It is now suggested to use Function components along with Hooks, which were added in React 16.8.

### **Files Structure:**



### index.js

```
import React from "react";
import ReactDOM from "react-dom";
import App from "../components/App";

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

### App.jsx

```
import React from "react";
import Heading from "../Heading";
import List from "../List";

function App() {
  return (
    <div>
      <Heading />
      <List />
      <List />
    </div>
  );
}

export default App;
```

### Heading.jsx

```
import React from "react";

function Heading() {
  return <h1>My Favourite Foods</h1>;
}

export default Heading;
```

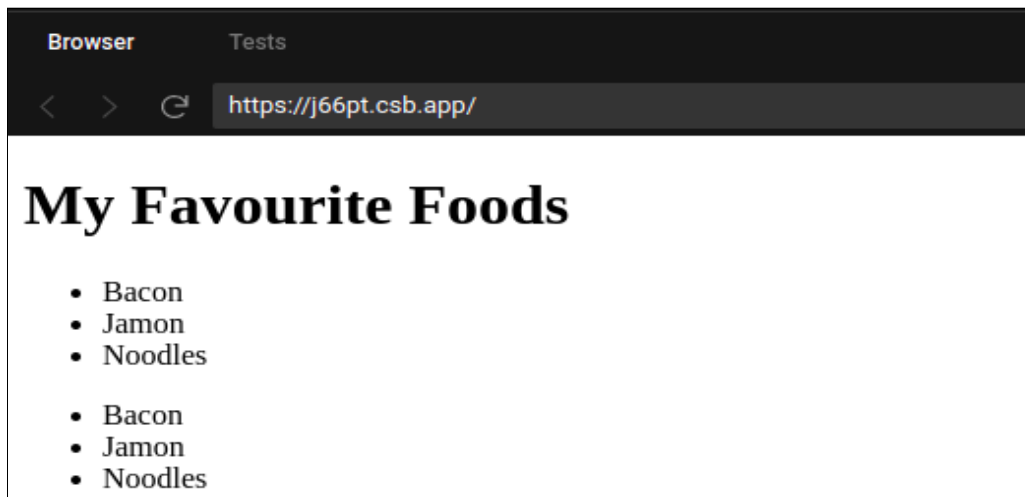
### List.jsx

```
import React from "react";

function List() {
  return (
    <ul>
      <li>Bacon</li>
      <li>Jamon</li>
      <li>Noodles</li>
    </ul>
  );
}

export default List;
```

### Output:



Code link : <https://codesandbox.io/s/react-components-j66pt?fontsize=14&file=/src/index.js>

## Javascript ES6 - Import, Export and Modules

This allows us to split large JS files into individual more manageable components.

**index.js** (call the functions with brackets)

```
import React from "react";
import ReactDOM from "react-dom";
import pi, { doublePi, triplePi } from "./math.js";

ReactDOM.render(
  <ul>
    <li>{pi}</li>
    <li>{doublePi()}</li>
    <li>{triplePi()}</li>
  </ul>,
  document.getElementById("root")
);
```

**math.js** (we can have one default export, but to export multiple functions we need to follow the below syntax)

```
const pi = 3.1415962;

function doublePi() {
  return pi * 2;
}

function triplePi() {
  return pi * 3;
}

export default pi;
export { doublePi, triplePi };
```

**Code link :**

<https://codesandbox.io/s/es6-importexport-modules-enyxr?fontsize=14&file=/src/index.js>

**Practice :**

<https://codesandbox.io/s/es6-importexport-practice-h1f0g?fontsize=14&file=/src/index.js>

## React Props:

- **props** stands for **properties**.
- Props are arguments passed into React components.
- Props are passed to components via HTML attributes.

HTML	Attributes
<pre>document.getElementById("email")  .id .placeholder .value</pre>	<pre>&lt;input   id="email"   placeholder="Your email"   value="angela@email.com" &gt;</pre>

Similar to HTML attributes we can define properties in react components.

React	Props
<pre>function Card(props) {   return &lt;div&gt;     &lt;h2&gt;{props.name}&lt;/h2&gt;     &lt;p&gt;{props.tel}&lt;/p&gt;     &lt;p&gt;{props.email}&lt;/p&gt;   &lt;/div&gt;; }</pre>	<pre>&lt;Card   name="Beyonce"   tel="+123456789"   email="b@beyonce.com" &gt;</pre>

### index.js

```
import React from "react";  
import ReactDOM from "react-dom";  
  
function Card(props) {  
  return (  
    <div>  
      <h2>{props.name}</h2>  
      <p>{props.tel}</p>  
      <p>{props.email}</p>  
    </div>  
  )  
}
```

```

    );
  }

ReactDOM.render(
  <div>
    <h1>My Contacts</h1>
    <Card
      name="Beyonce"
      tel="+123 456 789"
      email="b@beyonce.com"
    />
    <Card
      name="Jack Bauer"
      tel="+7387384587"
      email="jack@nowhere.com"
    />
  </div>,
  document.getElementById("root")
);

```

**Code link :** <https://codesandbox.io/s/react-props-gslmr?fontsize=14&file=/src/index.js>

**Practice :**

<https://codesandbox.io/s/react-props-practice-completed-c6fkx?fontsize=14&file=/src/index.js>

## React DEV TOOLS :

- React Developer Tools is a Chrome DevTools extension for the open-source React JavaScript library. It allows you to inspect the React component hierarchies in the Chrome Developer Tools.
- **Link :** <https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en>

## Mapping Data to Components :

- **map function** is used to handle arrays of objects in react components.
- It acts as a for loop to iterate over an array of objects.

**App.jsx** (used map function which acts as a for loop to iterate over the contacts array. )

```
import React from "react";
import Card from "../Card";
import contacts from "../contacts";

function createCard(contact) {
  return (
    <Card
      key={contact.id}
      name={contact.name}
      img={contact.imgURL}
      tel={contact.phone}
      email={contact.email}
    />
  );
}

function App() {
  return (
    <div>
      <h1 className="heading">My Contacts</h1>
      {contacts.map(createCard)}

      {/* <Card
        name={contacts[0].name}
        img={contacts[0].imgURL}
        tel={contacts[0].phone}
        email={contacts[0].email}
      />
      <Card
        name={contacts[1].name}
        img={contacts[1].imgURL}
        tel={contacts[1].phone}
        email={contacts[1].email}
      />*/}

    </div>
  );
}

export default App;
```

**contacts.js** (Array of object)

```
const contacts = [  
  {  
    id: 1,  
    name: "Beyonce",  
    imgURL:  
  
    "https://blackhistorywall.files.wordpress.com/2010/02/picture-device-indepe  
ndent-bitmap-119.jpg",  
    phone: "+123 456 789",  
    email: "b@beyonce.com"  
  },  
  {  
    id: 2,  
    name: "Jack Bauer",  
    imgURL:  
  
    "https://pbs.twimg.com/profile_images/625247595825246208/X3XLea04_400x400.j  
pg",  
    phone: "+987 654 321",  
    email: "jack@nowhere.com"  
  },  
];  
  
export default contacts;
```

**Code link :** <https://codesandbox.io/s/mapping-components-y6z4c?fontsize=14&file=/src/index.js>

**Practice :**

<https://codesandbox.io/s/mapping-components-practice-37h04?fontsize=14&file=/src/index.js>

## **Javascript ES6 Map/Filter/Reduce :**

**Map :** Create a new array by doing something with each item in an array.

```
var numbers = [3, 56, 2, 48, 5];  
function double(x) {  
  return x * 2;  
}  
const newNumbers = numbers.map(double);  
console.log(newNumbers);
```



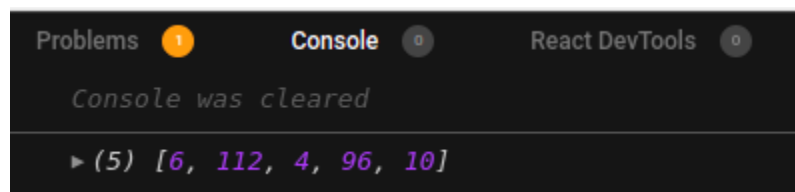
OR

```
var numbers = [3, 56, 2, 48, 5];
var newNumbers = [];
numbers.forEach(function (x) {
  newNumbers.push(x * 2);
});
console.log(newNumbers);
```

OR (Anonymous function inside map function)

```
var numbers = [3, 56, 2, 48, 5];
const newNumbers = numbers.map(function (x) {
  return x * 2;
});
console.log(newNumbers);
```

Output:

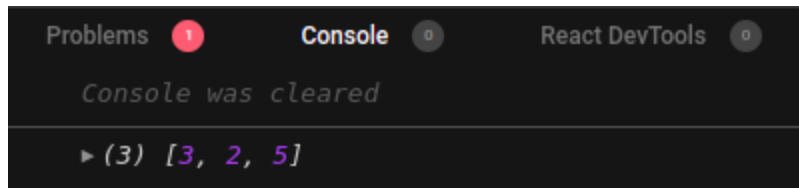


**Filter** : Create a new array by keeping the items that return true.

```
var numbers = [3, 56, 2, 48, 5];
const newNumbers = numbers.filter(function(num) {
  return num < 10;
});
console.log(newNumbers);
```

```
var numbers = [3, 56, 2, 48, 5];
var newNumbers = [];
numbers.forEach(function(num) {
  if (num < 10) {
    newNumbers.push(num);
  }
})
console.log(newNumbers);
```

Output:

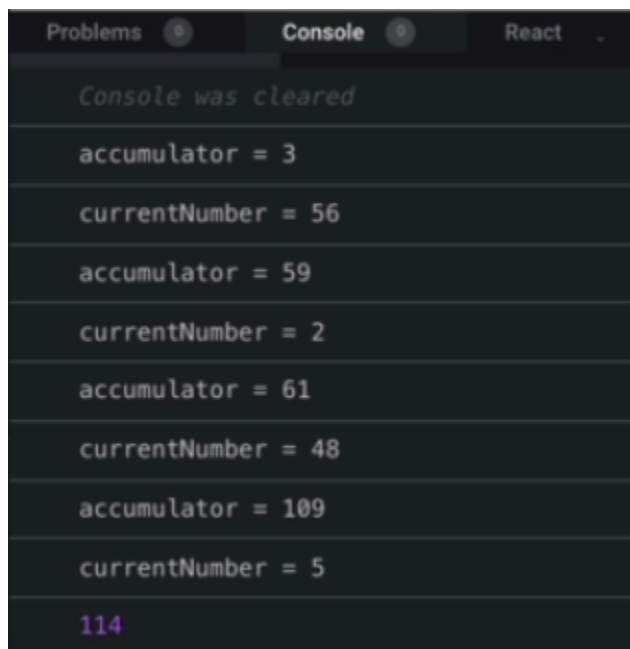


**Reduce** : Accumulate a value by doing something to each item in an array.

```
var numbers = [3, 56, 2, 48, 5];
var newNumber = numbers.reduce(function (accumulator, currentNumber) {
  console.log("accumulator = " + accumulator);
  console.log("currentNumber = " + currentNumber);
  return accumulator + currentNumber;
})
```

```
var numbers = [3, 56, 2, 48, 5];
var newNumber = 0;
numbers.forEach(function (currentNumber) {
  newNumber += currentNumber
})
```

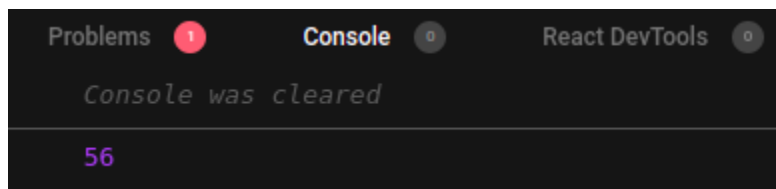
**Output :**



**Find** : find the first item that matches from an array.

```
var numbers = [3, 56, 2, 48, 5];
const newNumber = numbers.find(function (num) {
  return num > 10;
})
console.log(newNumber);
```

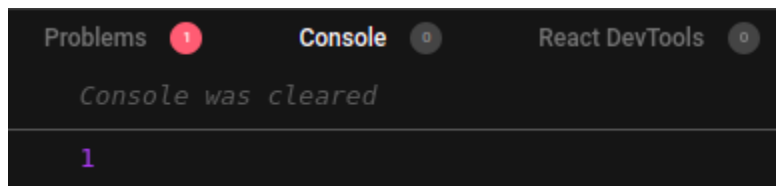
Output:



**FindIndex** : find the index of the first item that matches.

```
var numbers = [3, 56, 2, 48, 5];
const newNumber = numbers.findIndex(function (num) {
  return num > 10;
})
console.log(newNumber);
```

Output:



Code link : <https://codesandbox.io/s/mapfilterreduce-3sm6u?fontsize=14&file=/src/index.js>

## **Javascript ES6 Arrow Functions:**

Arrow functions allow us to write shorter function syntax:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "../components/App";

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

var numbers = [3, 56, 2, 48, 5];

////Map -Create a new array by doing something with each item in an array.
// const newNumbers = numbers.map(function (x) {
//   return x * 2;
// });

const newNumbers_Map = numbers.map((x) => x * 2);

/////Filter - Create a new array by keeping the items that return true.
// const newNumbers = numbers.filter(function(num) {
//   return num < 10;
// });

const newNumbers_Filter = numbers.filter((num) => num < 10);

//Reduce - Accumulate a value by doing something to each item in an array.
// var newNumber = numbers.reduce(function (accumulator, currentNumber) {
//   return accumulator + currentNumber;
// })

var newNumber_Reduce = numbers.reduce(
  (accumulator, currentNumber) => accumulator + currentNumber
);

/////Find - find the first item that matches from an array.
// const newNumber = numbers.find(function (num) {
//   return num > 10;
// })

const newNumber_Find = numbers.find((num) => num > 10);

/////FindIndex - find the index of the first item that matches.
// const newNumber = numbers.findIndex(function (num) {
//   return num > 10;
// })
```

```
const newNumber_FindIndex = numbers.findIndex((num) => num > 10);

console.log(newNumbers_Map);
console.log(newNumbers_Filter);
console.log(newNumber_Reduce);
console.log(newNumber_Find);
console.log(newNumber_FindIndex);
```

Code link : <https://codesandbox.io/s/es6-arrow-functions-forked-e4chh?file=/src/index.js>

## React Conditional Rendering with the Ternary operator & AND operator:

### Ternary Operator:

- The ternary operator is a simplified conditional operator like if / else.
- **Syntax:** condition ? <expression if true> : <expression if false>

CONDITION ? DO IF TRUE : DO IF FALSE

### AND Operator:

(EXPRESSION && EXPRESSION)

```
var x = 5;
```

```
(x > 3 && x < 7)
```

true true

(EXPRESSION && EXPRESSION)

```
var x = 1;
```

```
(x > 3 && x < 7)
```

false

## && in JS

```
(EXPRESSION && EXPRESSION)
```

```
(x > 3 && x < 7)
```

## && in React

```
CONDITION && EXPRESSION
```

```
true && EXPRESSION
```

```
false && EXPRESSION
```

Rendering Login Page and User Page depends on the value of isLoggedIn:

```
import React from "react";
import Login from "./Login";

var isLoggedIn = true;

const currentTime = new Date(2019, 12, 1, 9).getHours();
console.log(currentTime);

function App() {
  return (
    <div className="container">
      {/*Ternary Operator*/}
      {isLoggedIn ? <h1>Hello</h1> : <Login />}
      {/*AND Operator*/}
      {currentTime > 12 && <h1>Why are you still working?</h1>}
    </div>
  );
}

export default App;
```

**Code link :**

<https://codesandbox.io/s/conditional-rendering-ovu1v?fontsize=14&hidennavigation=1&theme=dark&file=/src/index.js>

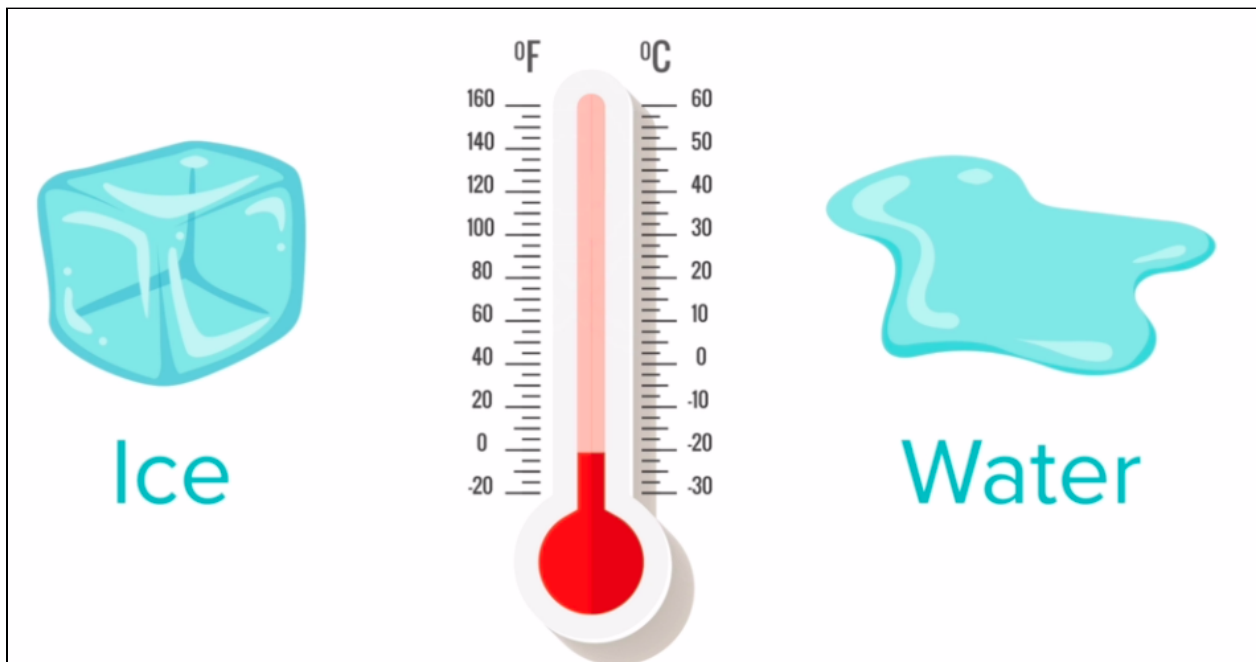
**Practice :** <https://codesandbox.io/s/conditional-rendering-practice-completed-forked-x14hss>

## React Hooks:

State is used to make the website more interactive. User Interface(UI) is a **function of State**.

$$UI = f(State)$$


Considering the analogy of ice and water, they are basically the same thing but their appearance depends on the state of temperature.



**Ice**


$$UI = f(-10)$$

**Water**


$$UI = f(60)$$

- Hooks allow function components to have access to state and other React features.
- Hooks can only be called inside React function components.
- Hooks can only be called at the top level of a component.
- With Hooks we can use class component features in functional components such as state, life cycle, pure component etc.

## 1. useState

The React useState Hook allows us to track state (data or properties) in a function component.

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

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

  function increase() {
    setCount(count + 1);
  }

  function decrease() {
    setCount(count - 1);
  }

  return (
    <div className="container">
      <h1>{count}</h1>
      <button onClick={decrease}>-</button>
      <button onClick={increase}>+</button>
    </div>
  );
}

export default App;
```



Code link : <https://codesandbox.io/s/useState-hook-completed-forked-texr67>

Practice : <https://codesandbox.io/s/useState-hook-practice-completed-t99f4>

## 2. useEffect

- The useEffect Hook allows you to perform side effects in your components.
- Some examples of side effects are: fetching data, directly updating the DOM, and timers.
- useEffect accepts two arguments. The second argument is optional.
  - **useEffect(<function>, <dependency>)**

Link :

-  React tutorial in Hindi #29 useEffect Hook in ReactJs
-  React tutorial in Hindi #30 useEffect with condition | part 2 | like component did mo...
- [React useEffect Hooks](#)



Use `setTimeout()` to count 1 second after initial render:

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom";

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

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  });

  return <h1>I've rendered {count} times!</h1>;
}

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

But wait!! It keeps counting even though it should only count once!

**useEffect** runs on every render. That means that when the count changes, a render happens, which then triggers another effect.

### Different ways to use `useEffect`:

#### a) No dependency passed:

```
useEffect(() => {
  //Runs on every render
});
```

#### b) An empty array:

```
useEffect(() => {
  //Runs only on the first render
}, []);
```

#### c) Props or state values:

```
useEffect(() => {
  //Runs on the first render
  //And any time any dependency value changes
}, [prop, state]);
```

## JavaScript ES6 Object & Array Destructuring:

The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

### data.js

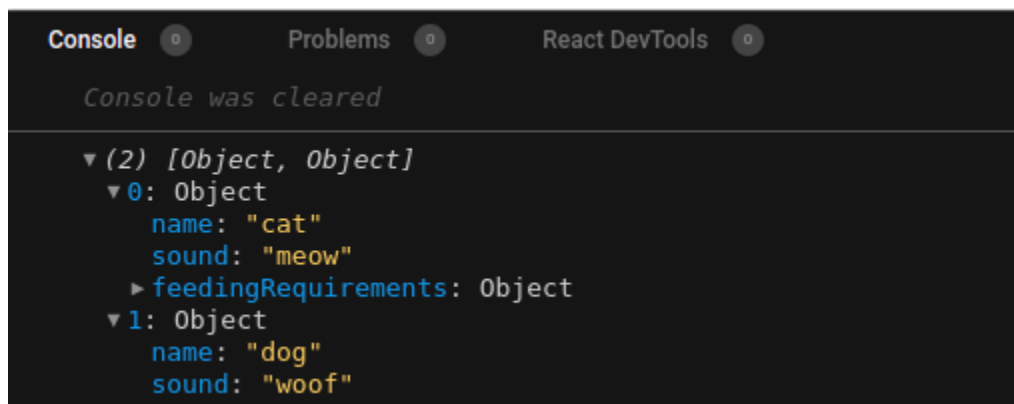
```
const animals = [  
  {  
    name: "cat",  
    sound: "meow",  
    feedingRequirements: {  
      food: 2,  
      water: 3  
    }  
  },  
  { name: "dog", sound: "woof" }  
];
```

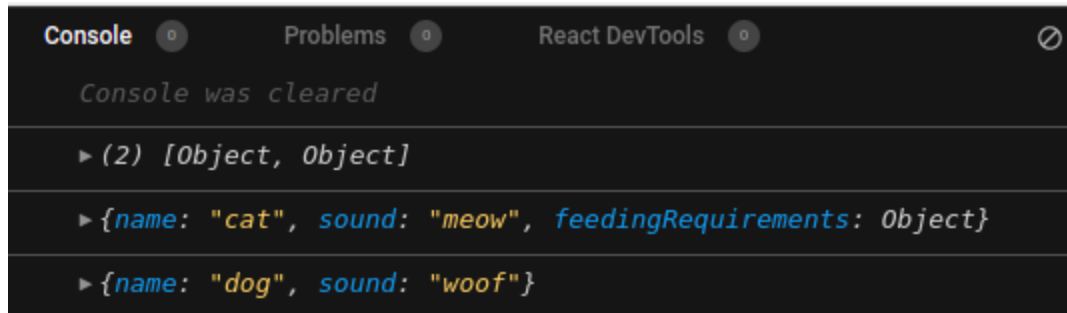
### index.js

#### Destructuring Arrays

```
import animals from "./data";  
  
//Destructuring Arrays  
console.log(animals);  
const [cat, dog] = animals;  
//var cat=animals[0] //Equivalent to this  
console.log(cat);  
console.log(dog);
```

### Outputs:





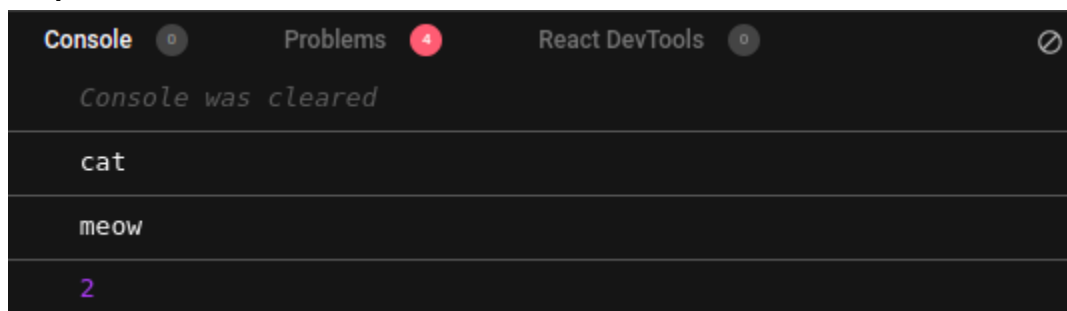
## Destructuring Objects

```
import animals from "./data";

const [cat, dog] = animals;

//Destructuring Objects
const { name, sound } = cat;
console.log(name); //Similar to cat.name
const { name: catName, sound: catSound } = cat;
console.log(catSound);
//const { name = "Fluffy", sound = "Purr" } = cat;
//If name is not defined on cat object then it will take this value.
const {
  feedingRequirements: { food, water }
} = cat;
//Nested objects
console.log(food);
```

## Output:



## Code link :

<https://codesandbox.io/s/es6-destructuring-zvjn9?fontsize=14&hiddenavigation=1&theme=dark>

## Event Handling in React:

- Just like HTML DOM events, React can perform actions based on user events.
- React has the same events as HTML: click, change, mouseover etc.
- React events are written in camelCase syntax:
- onClick instead of onclick.

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

function App() {
  const [headingText, setHeadingText] = useState("Hello");
  const [isMouseOver, setMouseOver] = useState(false);

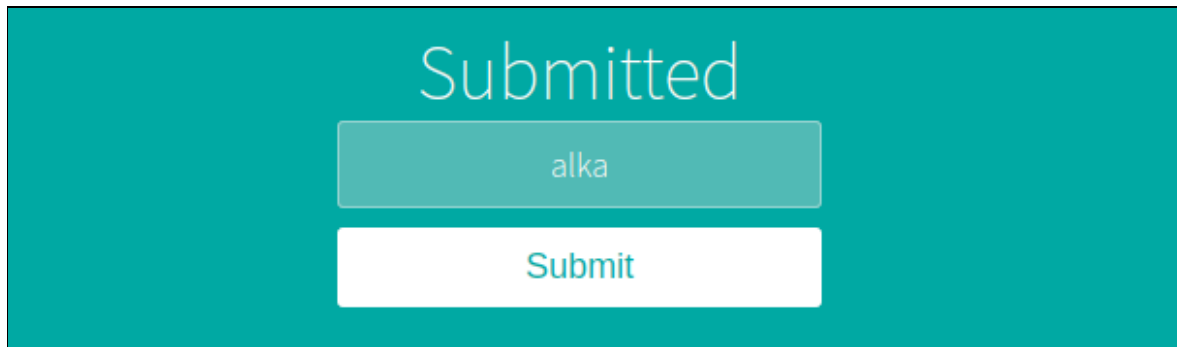
  function handleClick() {
    setHeadingText("Submitted");
  }

  function handleMouseOver() {
    setMouseOver(true);
  }

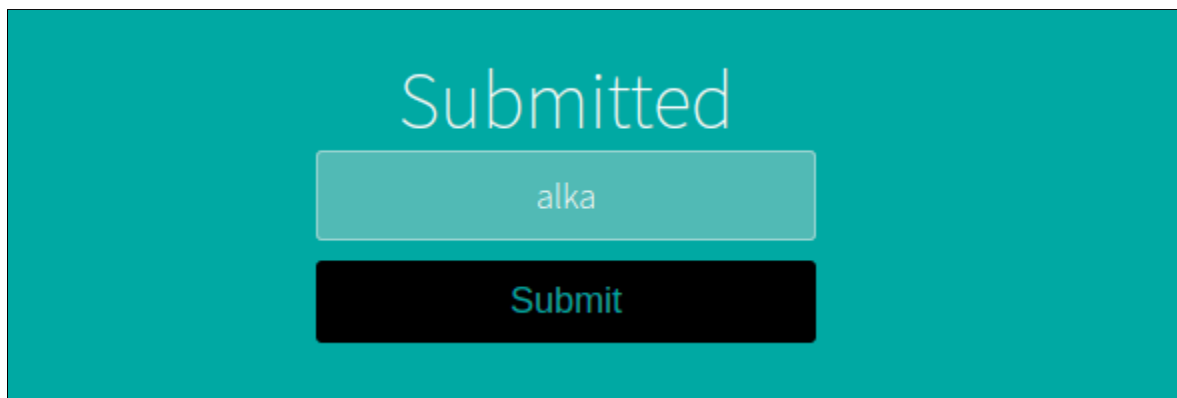
  function handleMouseOut() {
    setMouseOver(false);
  }

  return (
    <div className="container">
      <h1>{headingText}</h1>
      <input type="text" placeholder="What's your name?" />
      <button
        style={{ backgroundColor: isMouseOver ? "black" : "white" }}
        onClick={handleClick}
        onMouseOver={handleMouseOver}
        onMouseOut={handleMouseOut}
      >
        Submit
      </button>
    </div>
  );
}

export default App;
```



**OnClick** will change color to black



Code link : <https://codesandbox.io/s/event-handling-in-react-completed-forked-bgi3g9>

## React Forms:

- Just like in HTML, React uses forms to allow users to interact with the web page.
- In HTML, form data is usually handled by the DOM.
- In React, form data is usually handled by the components.
- When the data is handled by the components, all the data is stored in the component state.
- You can control changes by adding event handlers in the **onChange** attribute.

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

function App() {
  const [name, setName] = useState("");
  const [headingText, setHeading] = useState("");

  function handleChange(event) {
    console.log(event.target.value);
    setName(event.target.value);
  }
}
```

```

function handleClick(event) {
  setHeading(name);

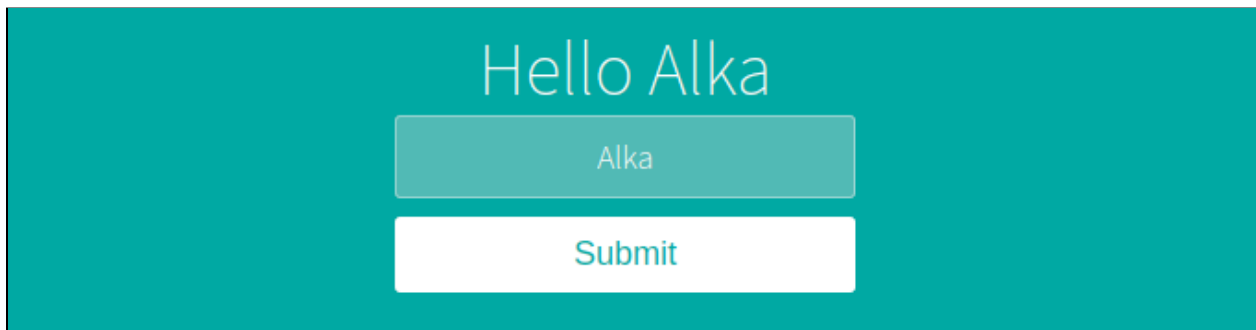
  event.preventDefault();
}

return (
  <div className="container">
    <h1>Hello {headingText}</h1>
    <form onSubmit={handleClick}>
      <input
        onChange={handleChange}
        type="text"
        placeholder="What's your name?"
        value={name}
      />
      <button type="submit">Submit</button>
    </form>
  </div>
);
}

export default App;

```

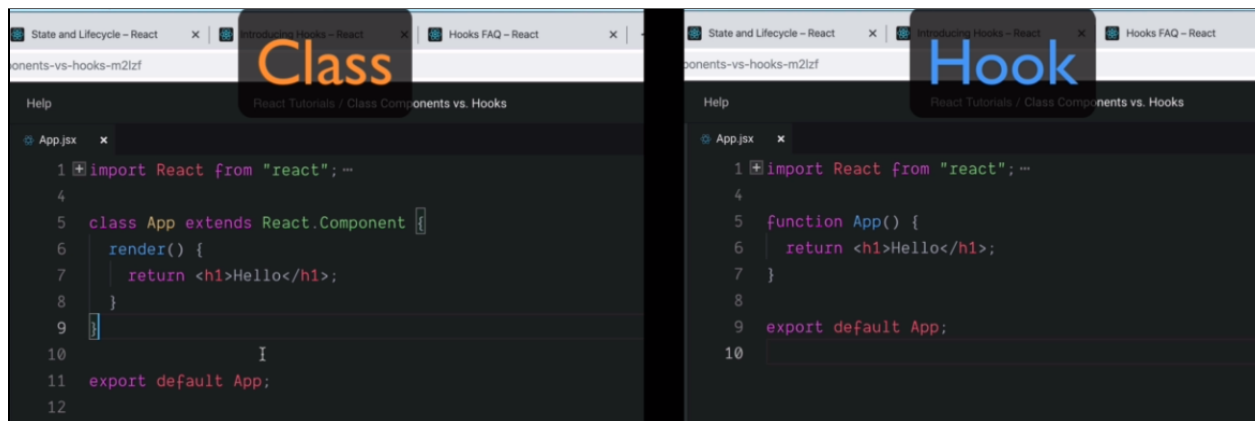
**Output:**



**Code link :** <https://codesandbox.io/s/react-forms-completed-forked-vm3cdf>

**Class Components vs Functional Components:**

Functional Component	Class Component
<ol style="list-style-type: none"> <li>1. A functional component is just a plain JavaScript function that accepts props as an argument and returns a React element.</li> <li>2. There is no render method used in functional components.</li> <li>3. React lifecycle methods (for example, componentDidMount) cannot be used in functional components.</li> </ol>	<ol style="list-style-type: none"> <li>1. A class component requires you to extend from React. Component and create a render function which returns a React element.</li> <li>2. It must have the render() method returning JSX (which is syntactically similar to HTML)</li> <li>3. React lifecycle methods can be used inside class components (for example, componentDidMount).</li> </ol>



## Changing Complex State:

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

function App() {
  const [contact, setContact] = useState({
    fName: "",
    lName: "",
    email: ""
  });

  function handleChange(event) {
    const { name, value } = event.target;

    setContact(prevValue => {
      if (name === "fName") {
        return {
          fName: value,
          lName: prevValue.lName,

```

```

        email: prevValue.email
    };
} else if (name === "lName") {
    return {
        fName: prevValue.fName,
        lName: value,
        email: prevValue.email
    };
} else if (name === "email") {
    return {
        fName: prevValue.fName,
        lName: prevValue.lName,
        email: value
    };
}
});
}
});
}

return (
    <div className="container">
        <h1>
            Hello {contact.fName} {contact.lName}
        </h1>
        <p>{contact.email}</p>
        <form>
            <input
                onChange={handleChange}
                value={contact.fName}
                name="fName"
                placeholder="First Name"
            />
            <input
                onChange={handleChange}
                value={contact.lName}
                name="lName"
                placeholder="Last Name"
            />
            <input
                onChange={handleChange}
                value={contact.email}
                name="email"
                placeholder="Email"
            />

```



```

        <button>Submit</button>
      </form>
    </div>
  );
}

export default App;

```

**Code link :** <https://codesandbox.io/s/changing-complex-state-practice-completed-forked-i6ts9i>

## **JavaScript ES6 Spread Operator :**

The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.

```

<!DOCTYPE html>
<html>

<body>

<script>
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne, ...numbersTwo];

```

```
document.write(numbersCombined);
</script>

</body>
</html>
```

**Output :** 1,2,3,4,5,6

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

function App() {
  const [contact, setContact] = useState({
    fName: "",
    lName: "",
    email: ""
  });

  function handleChange(event) {
    const { name, value } = event.target;

    setContact(prevValue => {
      return {
        ...prevValue,
        [name]: value
      };
    });
  }

  return (
    <div className="container">
      <h1>
        Hello {contact.fName} {contact.lName}
      </h1>
      <p>{contact.email}</p>
      <form>
        <input
          onChange={handleChange}
          name="fName"
          value={contact.fName}
          placeholder="First Name"
        />
        <input
```

```

        onChange={handleChange}
        name="lName"
        value={contact.lName}
        placeholder="Last Name"
      />
      <input
        onChange={handleChange}
        name="email"
        value={contact.email}
        placeholder="Email"
      />
      <button>Submit</button>
    </form>
  </div>
);
}

export default App;

```

**Code link :** <https://codesandbox.io/s/es6-spread-operator-completed-forked-w3jxqh>

## **React Context API and useContext hook**

▶ Context API and useContext hook tutorial - ReactJs

### **Global State and Lifting up the State**

- In React, sharing state is accomplished by moving it up to the closest common ancestor of the components that need it.
- This allows us to more easily share state among all of these components that need to rely upon it.

For Example : If we have 3 components in our App.

```

  A
 / \
B   C

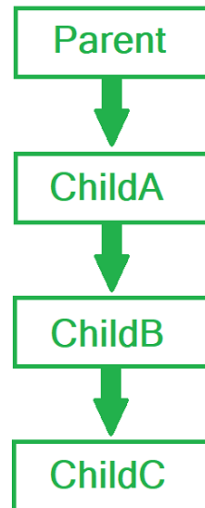
```

Where A is the parent of B and C. In this case, If there is some Data only in component B but, component C also wants that data. We know Component C cannot access the data because a component can talk only to its parent or child (Not cousins).

**To solve this**, we will Lift the state of component B and component C to component A.

## PropDrilling

Prop drilling is basically a situation when the same data is being sent at almost every level due to requirements in the final level. Here is a diagram to demonstrate it better. Data needed to be sent from Parent to ChildC.



## Context API

React's **Context API** simplifies the process of making data available to a large number of components, without having to manually pass that data through props at each level of your app's component tree.

### useContext Hook

“**useContext**” hook is used to create common data that can be accessed throughout the component hierarchy without passing the props down manually to each level. Context defined will be available to all the child components without involving “props”

## React Router

▶ React tutorial in Hindi #47 Routing Setup



## React API Integration - Fetch API

▶ React tutorial in Hindi #51 fetch API | GET method

## Local Storage

- ▶ Building a Todo List App Project with Local Storage in React JS & Hooks in 2021
- ▶ JavaScript Cookies vs Local Storage vs Session

## References :

1. <https://www.w3schools.com/REACT/DEFAULT.ASP>
2. <https://www.geeksforgeeks.org/differences-between-functional-components-and-class-components-in-react/>
3. Code Step By Step - YouTube React Tutorial
  - a.  React tutorial in Hindi #28 Hooks in ReactJs
  - b.  React tutorial in Hindi #30 useEffect with condition | part 2 | like component...
4. <https://www.udemy.com/course/the-complete-web-development-bootcamp/learn/lecture/17039602#overview>
5. <https://www.geeksforgeeks.org/lifting-state-up-in-reactjs/>
6. <https://www.geeksforgeeks.org/what-is-prop-drilling-and-how-to-avoid-it/>

---

---