

# 5.1 | React Deep dive

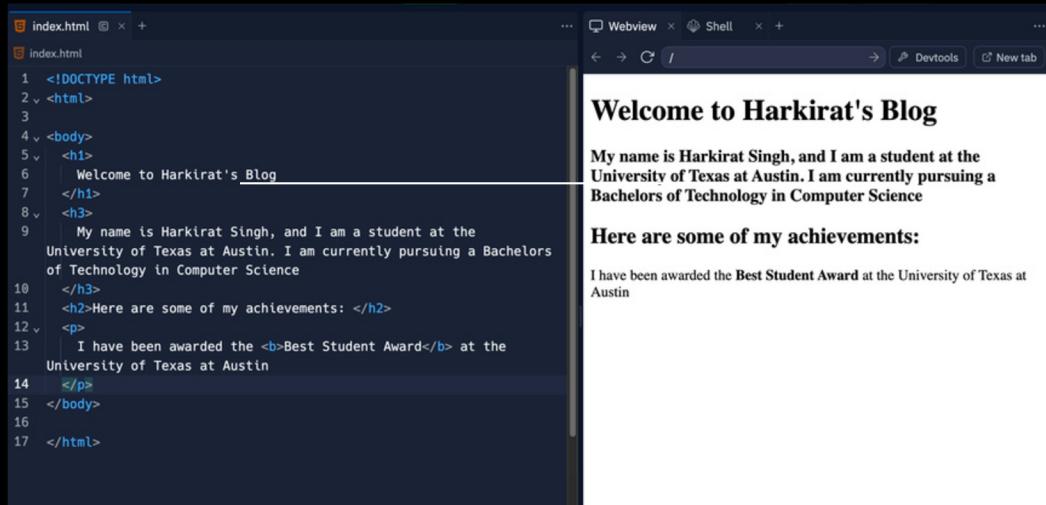
Understanding React from examples

# Jargon we'll learn today

Jsx, class vs className, static vs dynamic websites,  
State, components, re-rendering

# Why do you need React?

For static websites, you don't!



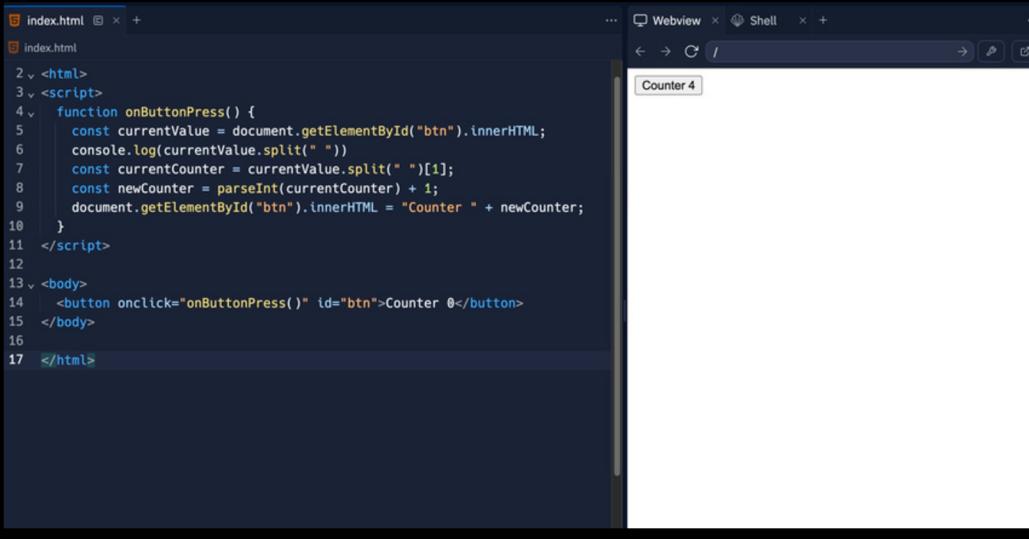
The image shows a split-screen view. On the left is a code editor window titled "index.html" containing the following HTML code:

```
1 <!DOCTYPE html>
2 <html>
3
4 <body>
5   <h1>
6     Welcome to Harkirat's Blog
7   </h1>
8   <h3>
9     My name is Harkirat Singh, and I am a student at the
10    University of Texas at Austin. I am currently pursuing a Bachelors
11    of Technology in Computer Science
12   </h3>
13   <h2>Here are some of my achievements: </h2>
14   <p>
15     I have been awarded the <b>Best Student Award</b> at the
16    University of Texas at Austin
17 </p>
18 </body>
19 </html>
```

On the right is a browser window titled "Webview" showing the rendered static website. The page title is "Welcome to Harkirat's Blog". The content includes a welcome message and a section titled "Here are some of my achievements:" which lists the "Best Student Award" achievement.

# Why do you need React?

For dynamic websites, these libraries make it easier to do DOM manipulation

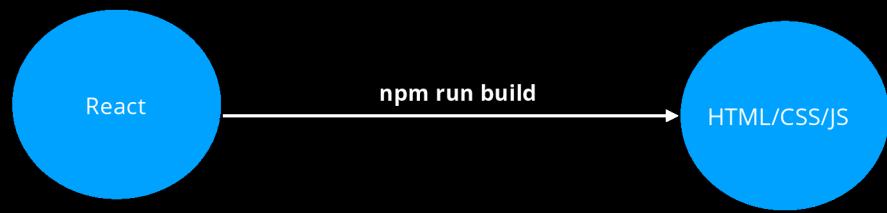


The image shows a code editor on the left and a browser window on the right. The code editor displays the contents of index.html:

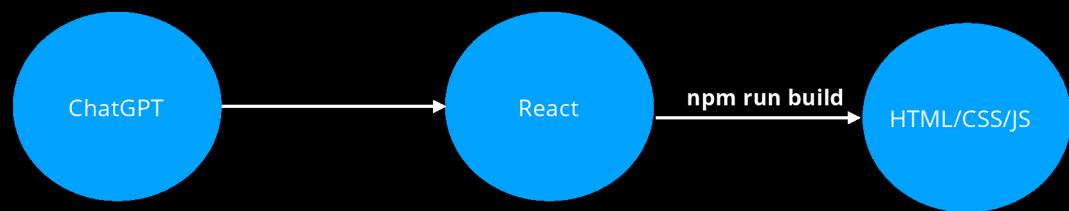
```
index.html
index.html
2 <html>
3 <script>
4   function onButtonPress() {
5     const currentValue = document.getElementById("btn").innerHTML;
6     console.log(currentValue.split(" "))
7     const currentCounter = currentValue.split(" ")[1];
8     const newCounter = parseInt(currentCounter) + 1;
9     document.getElementById("btn").innerHTML = "Counter " + newCounter;
10  }
11 </script>
12
13 <body>
14   <button onclick="onButtonPress()" id="btn">Counter 0</button>
15 </body>
16
17 </html>
```

The browser window shows a single button labeled "Counter 0".

React is just an easier way to write normal HTML/CSS/JS  
It's a new syntax, that under the hood gets converted to  
HTML/CSS/JS



Just how ChatGPT is an easier way to write code,  
React is an easier way to write HTML/CSS



## Why React?

People realised it's harder to do DOM manipulation the conventional way

There were libraries that came into the picture that made it slightly easy, but still for a very big app it's very hard (jQuery)

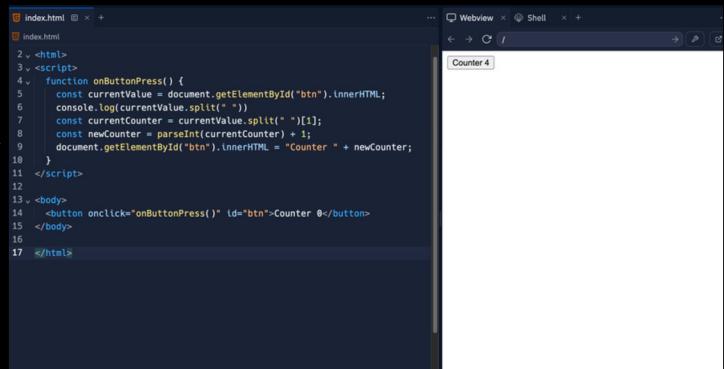
Eventually, VueJS/React created a new syntax to do frontends

Under the hood, the react compiler convert your code to HTML/CSS/JS

## Let's look at a simple example

### Problem with this approach

- 1.Too much code you have to write as the developer
- 2.As your app scales (todo app for eg), this gets harder and harder.



The screenshot shows a code editor with an open file named 'index.html'. The code contains a simple HTML structure with a script block that defines a function 'onButtonPress' which increments a counter and updates the button's innerHTML. To the right of the code editor is a browser window titled 'Counter 4' displaying the result of the code execution.

```
1 <!DOCTYPE html>
2 <html>
3   <script>
4     function onButtonPress() {
5       const currentValue = document.getElementById("btn").innerHTML;
6       console.log(currentValue.split(" "))
7       const currentCounter = currentValue.split(" ")[1];
8       const newCounter = parseInt(currentCounter) + 1;
9       document.getElementById("btn").innerHTML = "Counter " + newCounter;
10    }
11  </script>
12</html>
13<body>
14   <button onclick="onButtonPress()" id="btn">Counter 0</button>
15 </body>
16
17 </html>
```

<https://gist.github.com/hkirat/0c22122a9485d4d592b92677570e6bc8>

Some react jargon

## Some react jargon

To create a react app, you usually need to worry about two things

## Some react jargon

To create a react app, you usually need to worry about two things



State

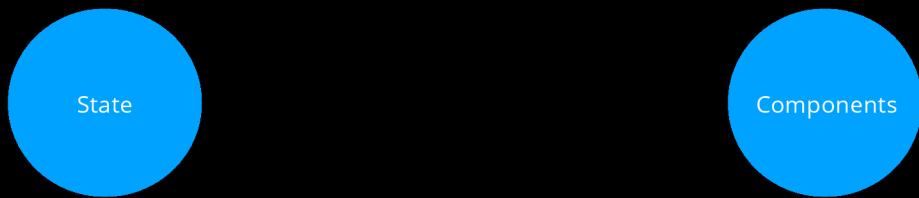


Components

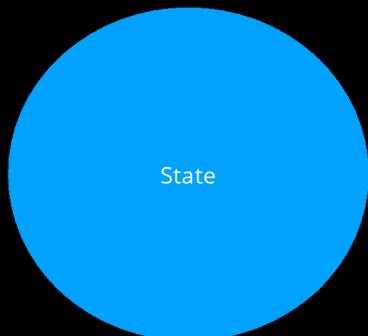
## Some react jargon

To create a react app, you usually need to worry about two things

Creators of frontend frameworks realised that all websites can effectively be divided into two parts



## State/Components/Re-rendering

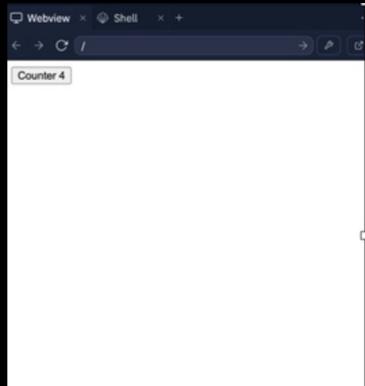


An object that represents the current **state** of the app

It represents the dynamic things in your app (things that change)

For example, the value of the counter

## State/Components/Re-rendering

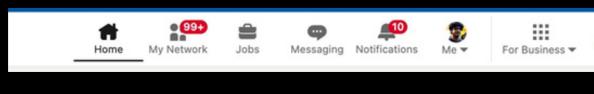


For the counter app, it could look something like this -

A screenshot of a code editor window titled "Untitled-1". The window has a dark purple background with three circular icons in the top-left corner. The main text area contains a single line of JSON-like code: "{ count: 1 }".

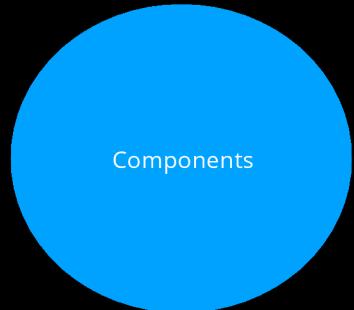
# State/Components/Re-rendering

For the LinkedIn Topbar, it could be something like this -



```
Untitled-1
{
  topbar: {
    home: 0,
    myNetwork: "99+",
    jobs: 0,
    messaging: 0,
    notifications: 10
  }
}
```

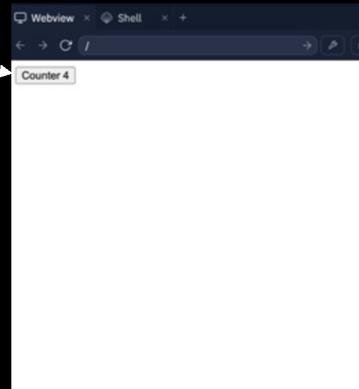
## State/Components/Re-rendering



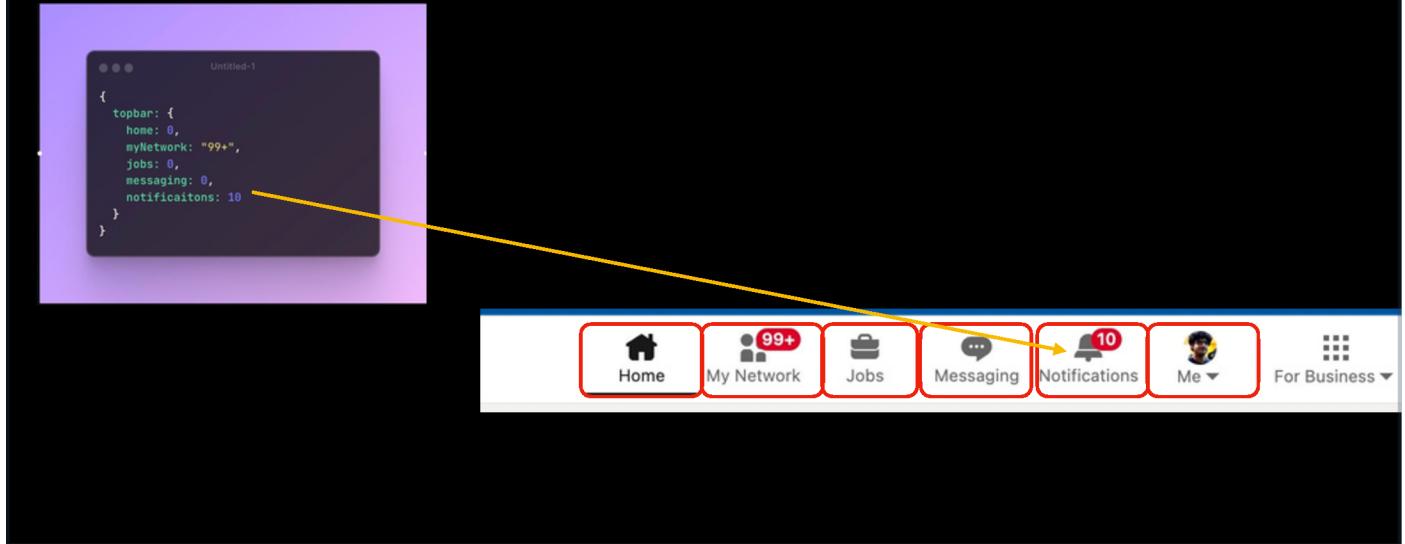
**How a DOM element should render, given a state  
It is a re-usable, dynamic, HTML snippet that changes given the state**

## State/Components/Re-rendering

This button is a component  
It takes the state (currentCount) as an input  
And is supposed to render accordingly



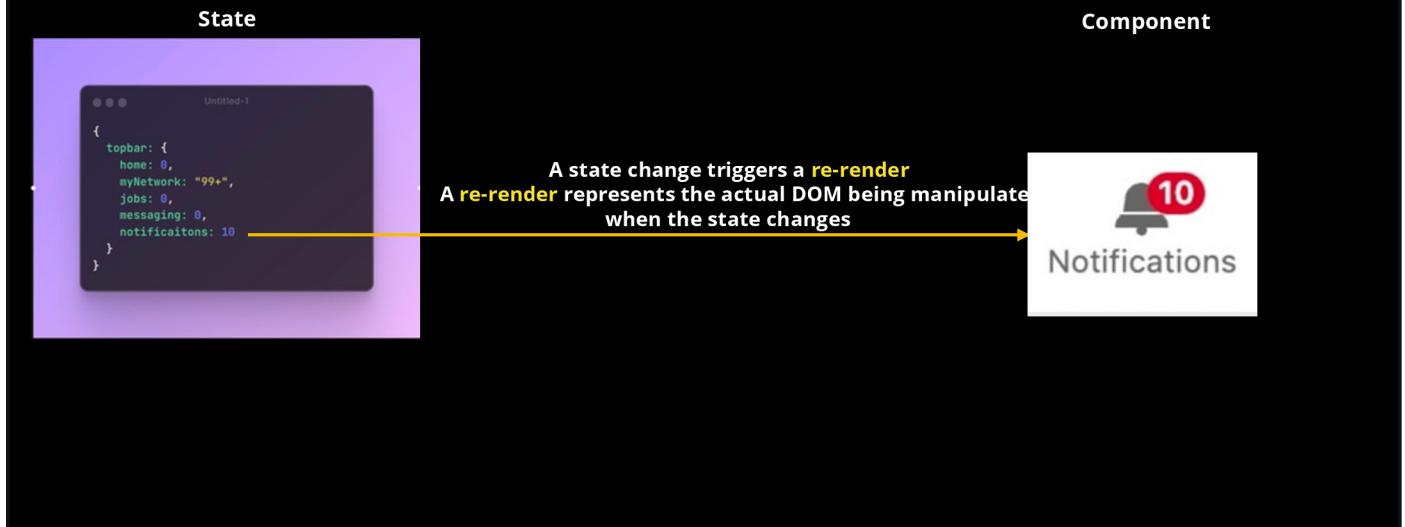
## State/Components/Re-rendering



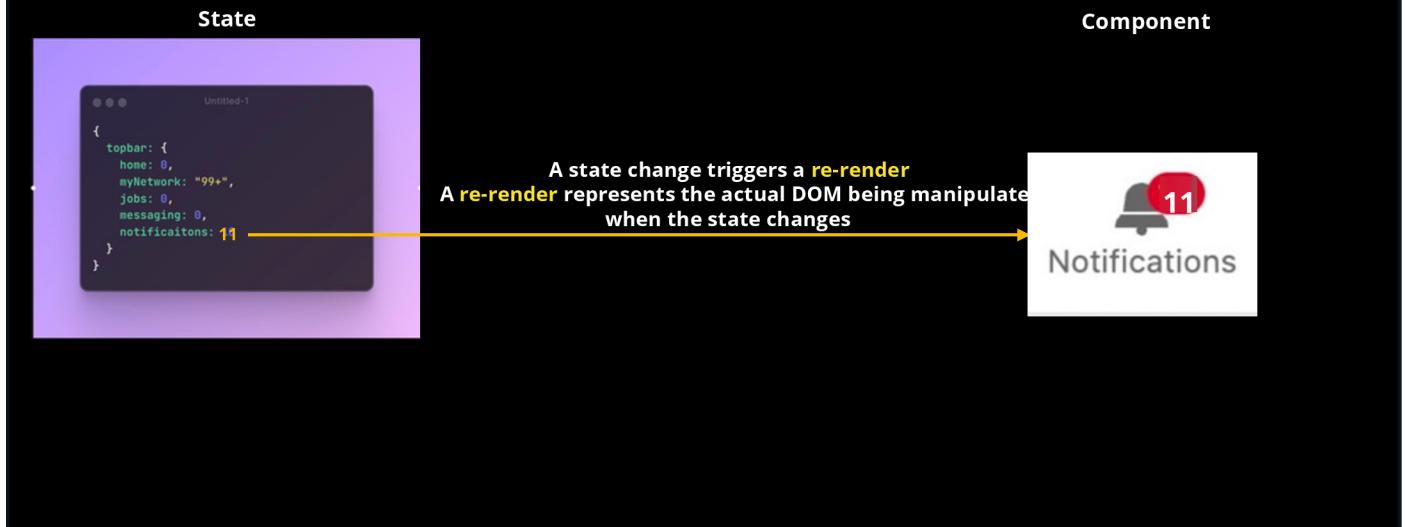
## State/Components/Re-rendering



## State/Components/Re-rendering



## State/Components/Re-rendering



## State/Components/Re-rendering

You usually have to define all your components once  
And then all you have to do is update the state of your app, React takes care of re-rendering your app

# Let's create a counter app using state/components

The screenshot shows a browser developer tools window. On the left, there is a code editor with the file 'index.html' open. The code contains JavaScript logic for a counter component. On the right, there is a preview pane showing a button labeled 'Counter 17'. Below the preview is a developer tools panel with tabs for Console, Elements, Network, Resources, Dom, and Settings. The Console tab is active, showing no output.

```
1 <!DOCTYPE html>
2 <html>
3   <body>
4     <div id="buttonParent">
5     </div>
6   </body>
7   <script>
8     let state = {
9       count: 0
10      }
11      function onButtonPress() {
12        state.count++;
13        buttonComponentReRender()
14      }
15      function buttonComponentReRender() {
16        document.getElementById("buttonParent").innerHTML = "";
17        const button = document.createElement("button");
18        button.innerHTML = `Counter ${state.count}`;
19        button.setAttribute("onclick", "onButtonPress()");
20        document.getElementById("buttonParent").appendChild(button);
21      }
22      buttonComponentReRender();
23    </script>
24  </body>
25 </html>
```

<https://gist.github.com/hkirat/c3d98735cec445e718b08f972dda7>

# Let's create a counter app using state/components

## 1. State initialisation

```
Untitled-1

<!DOCTYPE html>
<html>

<body>
  <div id="buttonParent">
  </div>
<script>
let state = {
  count: 0
}

function onButtonPress() {
  state.count++;
  buttonComponentReRender()
}

function buttonComponentReRender() {
  document.getElementById("buttonParent").innerHTML = "";
  const component = buttonComponent(state.count);
  document.getElementById("buttonParent").appendChild(component);
}

function buttonComponent(count) {
  const button = document.createElement("button");
  button.innerHTML = `Counter ${count}`;
  button.setAttribute("onclick", 'onButtonPress()');
  return button;
}

buttonComponentReRender();

</script>
</body>
</html>
```

# Let's create a counter app using state/components

## 2. Defining the button component



```
<!DOCTYPE html>
<html>
<body>
  <div id="buttonParent"></div>
</body>
</html>

<script>
  let state = {
    count: 0
  }

  function onButtonPress() {
    state.count++;
    buttonComponentReRender()
  }

  function buttonComponentReRender() {
    document.getElementById("buttonParent").innerHTML = "";
    const component = buttonComponent(state.count);
    document.getElementById("buttonParent").appendChild(component);
  }

  function buttonComponent(count) {
    const button = document.createElement("button");
    button.innerHTML = `Counter ${count}`;
    button.setAttribute("onclick", 'onButtonPress()');
    return button;
  }

  buttonComponentReRender();

</script>
</body>
</html>
```

The code editor shows a script block within an HTML document. Two specific sections of code are highlighted with red boxes and connected by arrows to the section header '2. Defining the button component'. The first highlighted section contains the definition of the `onButtonPress()` function, which increments the `state.count` and calls `buttonComponentReRender()`. The second highlighted section contains the definition of the `buttonComponent(count)` function, which creates a new button element, sets its innerHTML to 'Counter \${count}', and attaches an onclick event listener to call `onButtonPress()`.

# Let's create a counter app using state/components

The react library

```
Untitled-1

<!DOCTYPE html>
<html>
<body>
  <div id="buttonParent">
  </div>
<script>
  let state = {
    count: 0
  }

  function onButtonPress() {
    state.count++;
    buttonComponentReRender()
  }

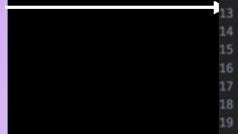
  function buttonComponentReRender() {
    document.getElementById("buttonParent").innerHTML = "";
    const component = buttonComponent(state.count);
    document.getElementById("buttonParent").appendChild(component);
  }

  function buttonComponent(count) {
    const button = document.createElement("button");
    button.innerHTML = `Counter ${count}`;
    button.setAttribute("onclick", 'onButtonPress()');
    return button;
  }

  buttonComponentReRender();

</script>
</body>
</html>
```

## The equivalent code in React looks like this



```
<!DOCTYPE html>
<html>
<body>
<div id="buttonParent">
</div>
<script>
let state = {
  count: 0
}

function onButtonPress() {
  state.count++;
  buttonComponentReRender()
}

function buttonComponentReRender() {
  document.getElementById("buttonParent").innerHTML = "";
  const component = buttonComponent(state.count);
  document.getElementById("buttonParent").appendChild(component);
}

function buttonComponent(count) {
  const button = document.createElement("button");
  button.innerHTML = `Counter ${count}`;
  button.setAttribute("onclick", `onButtonPress()`);
  return button;
}

buttonComponentReRender();
</script>
</body>
</html>
```

```
c > App.jsx > ...
1 import React from 'react'
2
3 function App() {
4   const [count, setCount] = React.useState(0)
5
6   return (
7     <div>
8       | <Button count={count} setCount={setCount}></Button>
9     </div>
10  )
11 }
12
13 function Button(props) {
14   function onButtonClick() {
15     props.setCount(props.count + 1);
16   }
17   return <button onClick={onButtonClick}>Counter {props.count}</button>
18 }
19
20 export default App
21 |
```

The equivalent code in React looks like this

```
c > App.jsx > ...
1  import React from 'react'
2
3  function App() {
4    const [count, setCount] = React.useState(0)
5
6    return (
7      <div>
8        <Button count={count} setCount={setCount}></Button>
9      </div>
10     )
11   }
12
13  function Button(props) {
14    function onButtonClick() {
15      props.setCount(props.count + 1);
16    }
17    return <button onClick={onButtonClick}>Counter {props.count}</button>
18  }
19
20  export default App
21 |
```

Lets start small, and then build up to this app

# The equivalent code in React looks like this

Lets start with a simple button component

```
App.jsx > Button
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {
  function onButtonClick() {
    props.setCount(count + 1);
  }

  return React.createElement(
    'button',
    { onClick: onButtonClick },
    `Counter ${props.count}`
  );
}

export default App
```

<https://gist.github.com/hkirat/db15d13b42c906269b3114efb96d820f>

The equivalent code in React looks like this

Defining Button component

```
App.jsx > Button
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {
  function onButtonClick() {
    props.setCount(count + 1);
  }

  return React.createElement(
    'button',
    { onClick: onButtonClick },
    `Counter ${props.count}`
  );
}

export default App
```

<https://gist.github.com/hkirat/8801c2cfad70853decd0ad1759d4e63c>

# The equivalent code in React looks like this

## Defining Button component

The image shows a comparison between two code snippets. The left snippet is a traditional JavaScript function-based approach, while the right snippet is a modern React component definition.

**Left Snippet (Traditional JS):**

```
<!DOCTYPE html>
<html>
<body>
  <div id="buttonParent"></div>
  <script>
    let state = {
      count: 0
    }

    function onButtonPress() {
      state.count++;
      buttonComponentReRender();
    }

    function buttonComponentReRender() {
      document.getElementById("buttonParent").innerHTML = "";
      const component = buttonComponent(state.count);
      document.getElementById("buttonParent").appendChild(component);
    }

    function buttonComponent(count) {
      const button = document.createElement("button");
      button.innerHTML = `Counter ${count}`;
      button.setAttribute("onclick", "onButtonPress()");
      return button;
    }

    buttonComponentReRender();
  </script>
</body>
</html>
```

**Right Snippet (React Component):**

```
App.jsx > ⚡ Button
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {

  function onButtonClick() {
    props.setCount(count + 1);
  }

  return React.createElement(
    'button',
    { onClick: onButtonClick },
    `Counter ${props.count}`
  );
}

export default App
```

<https://gist.github.com/hkirat/8801c2cfad70853decd0ad1759d4e63c>

# The equivalent code in React looks like this

Triggering re-render

```
@@@ Untitled-1

<!DOCTYPE html>
<html>

<body>
<div id="buttonParent">
</div>
<script>
let state = {
  count: 0
}

function onButtonPress() {
  state.count++;
  buttonComponentReRender();
}

function buttonComponentReRender() {
  document.getElementById("buttonParent").innerHTML = "";
  const component = buttonComponent(state.count);
  document.getElementById("buttonParent").appendChild(component);
}

function buttonComponent(count) {
  const button = document.createElement("button");
  button.innerHTML = `Counter ${count}`;
  button.setAttribute("onclick", "onButtonPress()");
  return button;
}

buttonComponentReRender();

</script>
</body>
</html>
```

```
App.jsx > ⚡ Button
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {
  function onButtonClick() {
    props.setCount(count + 1);
  }

  return React.createElement(
    'button',
    { onClick: onButtonClick },
    `Counter ${props.count}`
  );
}

export default App
```

<https://gist.github.com/hkirat/8801c2cfad70853decd0ad1759d4e63c>

# The equivalent code in React looks like this

Jsx syntax is a cleaner way to write components

```
App.jsx > Button
import React from 'react'

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

  return (
    <div>
      <Button count={count} setCount={setCount}></Button>
    </div>
  )
}

function Button(props) {
  function onButtonClick() {
    props.setCount(count + 1);
  }

  return React.createElement(
    'button',
    { onClick: onButtonClick },
    `Counter ${props.count}`
  );
}

export default App
```

```
src > App.jsx > ...
1   import React from 'react'
2
3   function App() {
4     const [count, setCount] = React.useState(0)
5
6     return (
7       <div>
8         <Button count={count} setCount={setCount}></Button>
9       </div>
10    )
11
12    function Button(props) {
13      function onButtonClick() {
14        props.setCount(props.count + 1);
15      }
16
17      return <button onClick={onButtonClick}>Counter {props.count}</button>
18    }
19
20  export default App
21 |
```

<https://gist.github.com/hkirat/8801c2cfad70853decd0ad1759d4e63c>

The equivalent code in React looks like this

**What Is jsx**

JSX stands for JavaScript XML. It is a syntax extension for JavaScript, most commonly used with React, a popular JavaScript library for building user interfaces. JSX allows you to write HTML-like code directly within JavaScript. This makes it easier to create and manage the user interface in React applications.

**<https://gist.github.com/hkirat/dcc85803a20639826bf8f64c6be24a31>**