Day 1 Session 2:

Step 1: JSX to add many HT ML elements and nest them to create complex Pages

Create a new file and Add App2 component

Edit index.js

Type the code from scratch

```
import React from 'react'
import './App.css'


        function App2() {
           return (
              <div className='container'>
                 <h1>Welcome to VIVA</h1>
                 <p>Let's Party</p>
                 <ul>
                    <li>
                       <button>
```

```
    </button>
</li>
<li>
    <button>
```

```
</span>
                </button>
            </li>
            <li>
                <button>
```

```
            </span>
                        </button>
                    </li>
                </ul>
            </div>
        );
    }
    export default App2
```

Style the elements by editing App.css

```
    .container {
      display: flex;
      flex-direction: column;
      align-items: center;
    }
    button {
      font-size: 2em;
      border: 0;
      padding: 0;
      background: none;
      cursor: pointer;
    }
    ul {
      display: flex;
      padding: 0;
    }
    li {
      margin: 0 20px;
      list-style: none;
      padding: 0;
    }
```

You've now worked with several JSX elements that look like regular HTML. You've added classes, ids, and aria tags, and have worked with data as strings and variables. But React also uses attributes to define how your elements should respond to user events.

Step 2 — Adding Events to Elements

In this step, you'll add events to elements using special attributes and capture a click event on a button element. You'll learn how to capture information from the event to dispatch another action or use other information in the scope of the file. Now that you have a basic page with information, it's time to add a few events to it. There are many event handlers that you can add to HTML elements. React gives you access to all of these. Since your JavaScript code is coupled with your markup, you can quickly add the events while keeping your code well-organized. To start, add the onclick event handler. This lets you add some JavaScript code directly to your element rather than attaching an event listener

```
import React from 'react'

import './App.css'


function App2() {

  return (

    <div className='container'>

      <h1>Welcome to VIVA</h1>

      <p>Let's Party</p>

      <ul>

        <li>

          <button onClick={event => alert(event.target.id)}>
```

```
    </button>
</li>
<li>
    <button onClick={event => alert(event.target.id)}>
```

```
            </span>
                </button>
            </li>
            <li>
                <button onClick={event => alert(event.target.id)}>
```

```
</span>
                </button>
              </li>
            </ul>
          </div>
      );
    }
    export default App2
```

Since this is JSX, you camelCased onclick , which means you added it as o nClick . This onClick attribute uses an anonymous function to retrieve information about the item that was clicked. You added an anonymous arrow function that will get the event from the clicked button, and the event will have a target that is the element. The information you need is in the id attribute, which you can access with event.target.id . You can trigger the alert with the alert() function.

Step 3 — Generalization of Above Code

You can reduce a duplication by declaring the function once and passing it to each onClick action. Since the function does not rely on anything other than inputs and outputs, you can declare it outside the main component function. In other words, the function does not need to access the scope of the component. The advantage to keeping them separate is that your component function is slightly shorter and you could move the function out to a separate file later if you wanted to.

```
import React from 'react'
import './App.css'


function App2() {
  const emojifn=event => alert(event.target.id);
  return (
    <div className='container'>
      <h1>Welcome to VIVA</h1>
      <p>Let's Party</p>
      <ul>
        <li>
          <button onClick={emojifn}>
```

```
    </button>
</li>
<li>
    <button onClick={emojifn}>
```

```
</span>
                </button>
              </li>
              <li>
                <button onClick={emojifn}>
```

```
</span>
                </button>
              </li>
            </ul>
          </div>
    );
}
export default App2
```

Step 4 — Mapping Over Data to Create Element

In this step, you'll move beyond using JSX as simple markup. You'll learn to combine it with JavaScript to create dynamic markup that reduces code and improves readability. You'll refactor your code into an array that you will loop over to create HTML elements. JSX doesn't limit you to an HTML-like syntax. It also gives you the ability to use JavaScript directly in your markup. You tried this a little already by passing functions to attributes. You also used variables to reuse data.

```
import React from 'react';
import './App.css';


const displayEmojiName = event => alert(event.target.id);
const emojis = [
 {
   emoji: './public/p1.jpg',
   name: "camera"
 },
 {
```

```
  name: "party popper"
},
{
```

```
    name: "woman dancing"
  }
];

function App() {
  const greeting = "greeting";
  const displayAction = false;
  return(
    <div className="container">
      <h1 id={greeting}>Hello, World</h1>
      {displayAction && <p>I am writing JSX</p>}
      <ul>
        {
          emojis.map(emoji => (
            <li key={emoji.name}>
              <button
                onClick={displayEmojiName}
              >
                                    <span role="img" src={emoji.emoji} aria-label={emoji.name}
id={emoji.name}>{emoji.emoji}</span>
              </button>
            </li>
          ))
        }
      </ul>
    </div>
  )
}
export default App;
```

Step 5: