Components :

As we know that components are reusable UI elements. And in normal programming, functions help us to put reusable piece of code at one place.

Now in a react code base if we have a function that returns some kind of JSX, we will call that a `component` or more specifically a functional component.
Why a functional component ?
Because this component has been made using functions.
Is there any other way to make components ? yes, we can make class components also.

Functional Component :

Writing functional component is easy, we use the function keyword and then name of the function. Name of the function is technically name of the component. This component must return some JSX to be called as component.

```
function APP() {
return(
<div>
      <h1> Hello,world </h1>
</div>
);
}
export default APP;          // it is accssible outside of the function
```

Now we know how to define it , but how to call it ?

Calling a component :

Component are functions and generally we call functions like this:

```
App( ... );
```

But because we are using JSX, it provides an alternative JSX compatible way to call out functional components.

```
<App />
```

The above JSX syntax calls the App component.

Note:

Name of the component must start with a capital letter else it will not work properly. The custom components must be in PascalCase and inbuilt HTML elements based components like `div` `h1,h2` etc should be in lowercase.
We should keep the file name also PascalCase.

Example:

The below code will not work:

```
import customComponent from "./CustomComponent";
function App() {
return (
<div>

<h1>Hello, world!</h1>
<customComponent />
<customComponent />
<customComponent />
<customComponent />
<customComponent />

</div>

);
}

export default App; // To make sure other files can access our APp function
```

We will get the following error in the browser:

`Warning: <customComponent /> is using incorrect casing. Use PascalCase for` React components, or lowercase for HTML elements.

Installation of tailwind in react project

1. Setup a new vite project
2. Inside the project install tailwindcss, postcss and autoprefixer

```
npm install -D tailwindcss postcss autoprefixer
```

3. Setup tailwind css config and post css config file

```
npx tailwind init -p
```

4. The above command will create the config files.
5. Go to your index.css file and add tailwind directives to enable tailwind styling classes in the project

```
/* index.css */*
@tailwind base;
@tailwind components;
@tailwind utilities;
```

`

6. The above code will add tailwind styling everywhere in our project because index.css is the root css styling that's applied everywhere
7. And that's it, tailwind is ready to use.
8. Now to use tailwind css in any component we need to add tailwind classes. But JSX is a part of JS, hence we cannot use class keyword in the JSX syntax to mimic html attribute class . So, in JSX instead of class we write className attribute to html element components.

```
function App() {

  return (

    <div>

      <h1 className="font-semibold text-3xl">Hello World!</h1>

    </div>
```

```
  );

}

export default App;
```

More details on JSX

In a JSX based component, if we have a single JSX to be returned, then that is directly done

```
function Button() {

    return  <button>Click Me</button>;

}

export default Button;
```

But for a multiple component, we need to wrap the JSX in a pair of parenthesis and then return it .

```
function Button() {
      return (
           <button>
             Click me
           </button>
      );

}
export default Button;
```

In a JSX component, there should be only one parent HTML element returned, i.e. we can only return one single element from a JSX component, if we want to return multiple elements then we need to wrap those multiple elements in a single parent element.
The below code will not work as it is returning multiple elements:

```
function Button() {

    return (
```

```
        <button>Click Me</button>

        <p>Click</p>

    );



}

export default Button;
```

But if we wrap the button and p tag in a common parent and then return it, it will work.

```
function Button() {

    return (
        <div>

        <button>Click Me</button>
        <p>Click</p>

        </div>

    );

}

export default Button;
```

Adding your button and P tag to single parent `div` might hamper your CSS. So, that where React Fragment comes into the picture

React Fragment

- Because of the constraint that we can only return a single parent from JSX component, we end up adding more wrapper elements on the ui, which might not be required. For example, the div wrapping button and p tag is an extra element. For a very complex UI, there can be many many components, and if each component return something irrelevant then we have a lot of non-required elements.

- To stop this we can use React fragments, fragments can be used to wrap the elements in a single parent, without adding any extra element on the html.

How to create fragment ?

```
function Button() {
    return (
      <>
           <button>Click Me</button>
            <p>Click</p>
     </>

    );
}
export default Button;
```

Another way to add fragment is to use `React.Fragment` component.

```
import React from "react";
function Button() {
    return (
      <React.Fragment>
           <button>Click Me</button>
            <p>Click</p>
     </React.Fragment>
    );
}
export default Button;
```

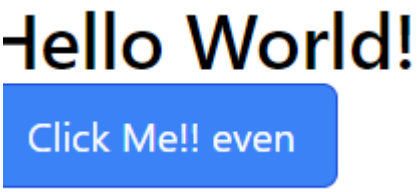
And with any one of the two used, we will still get no extra wrapper elements on the UI.

JSX curlies:

In JSX we can write some JS, which will be evaluated on the runtime and the return value is showed on the UI. To do this, we have to wrap out JS expression in a pair of curly braces called as JSX curlies.

```
import React from "react";
function Button() {
    return (
           <button

           className="px-4 py-2 bg-blue-500 border border-blue-700 text-white
hover:bg-blue-700

           hover:border-blue-900 rounded-md transition-all "
             >

               Click Me!!  {isEvenOrOdd(2)}

           </button>

    );

    function isEvenOrOdd(num) {

        return num % 2 == 0 ? "even" : "odd";

    }

}

export default Button;
```

OUTPUT:



This is going to show even or odd on the UI because that's the return values after evaluating the num.
For any kind of reusability we need this type of curiles we want
{isEvenOrOdd(3)} it gives odd

How to make functional component more alive ??

Props

props are property that we are going to pass as an argument to your JSX component

Currently our components have some static ui, there are not expecting any inputs. But in real function we have the logic confined and then we pass some inputs which is processed on the function and we get an output.
To pass some inputs to our components which they can consume we use something called as `props`.

```
import React from "react";
function Button({text, buttonType}) {

    return (

            <button

            type={buttonType}

            className="px-4 py-2 bg-blue-500 border border-blue-700 text-white
hover:bg-blue-700

            hover:border-blue-900 rounded-md"

            >
                 {text}

            </button>

    );
}

export default Button;
```

```
import Button from "./Components/Button/Button";

function App() {

  return (

    <div>

      <h1 className="font-semibold text-3xl">Hello World!</h1>

      <Button text="primary" buttonType="submit"/>
```

```
      <Button text="secondary" buttonType="button"/>

    </div>

  );

}

export default App;
```

So, here in the Button component, we have destructured the props object and fetched text and button type prop. We can use these props by wrapping them in JSX curlies as they need to be evaluated during runtime. (this is suitable and understandble way to write props)
If we don't want to destructure the props, then we can expect a props object directly as parameter and then use properties inside it.

```
import React from "react";
function Button(props) {

    return (

            <button

            type={props.buttonType}

            className="px-4 py-2 bg-blue-500 border border-blue-700 text-white
hover:bg-blue-700

            hover:border-blue-900 rounded-md"

            >

                 {props.text}

            </button>

    );



}

export default Button;
```

But how do we pass the values of these props ? we can pass the values from the call site of component in the form of `key=value` format which is very similar to that of HTML attributes but with just JSX.

```
<Button text="Secondary" buttonType="submit" />
```

So here text prop gets a `Secondary` value and type prop gets a `Submit value