COMPONENT TYPES

Stateful and Stateless Components

In a component, state is data we import — typically to show the user — that is subject to change. It could change because the database we're getting from may be updated, the user modified it — there are so many reasons that data changes

Stateful and stateless components have many different names.

They are also known as:
– Container vs Presentational components
– Smart vs Dumb components

The literal difference is that one has state, and the other doesn't. That means the stateful components are keeping track of changing data, while stateless components print out what is given to them via props, or they always render the same thing.

Stateful/Container/Smart component:
example:

```
class Main extends Component {
 constructor() {
   super()
   this.state = {
     books: []
   }
 }
 render() {
   return <BooksList books={this.state.books} />;
 }
}
```

Stateless/Presentational/Dumb component:

```
const BooksList = ({books}) => {
 return (
   <ul>
     {books.map(book => {
       return <li>book</li>
     })}
   </ul>
 )
}
```

Example

```
import React from 'react';
import Text from './Test';
const App =() =>{
   const [count,setCount] =React.useState(0);
   return (
      <div>
        <center>
        <h3>Count :{count}</h3>
        <button onclick={()=> setCount(count+1)}>Increment</button>
        <Test count={count} />
        </center>
      </div>
   )
}
export default App
```

```
import React from 'react';
const Test =(props) => {
   return (
      <div>
         <center>
            <h3>Count : {props.count} from stateLess component </h3>
         </center>
      </div>
   )
}
export default Test
```

PRESENTATIONAL AND CONTAINER COMPONENTS

In react the components are divided into two categories: presentational components and container components.

presentational component
Presentational components are mostly concerned with generating some markup to be outputted.

They don’t manage any kind of state, except for state related to the presentation

```
const Users = props =>
     (<ul>
       {props.users.map(user =>
         (<li>{itr}</li>    ))
       }
     </ul>)
```

container components
Container components are mostly concerned with the “backend” operations.

They might handle the state of various sub-components. They might wrap several presentational components. They might interface with Redux

```
class UsersContainer extends React.Component {
  constructor() {
    this.state = {
      users: []
    }
  }

  componentDidMount() {
    axios.get('/users').then(users =>
      this.setState({ users: users }))
    )
  }

  render() {
    return <Users users={this.state.users} />
  }
}
```

pure components
A React component is considered pure if it renders the same output for the same state and props. For this type of class component, React provides the `PureComponent` base class. Class components that extend the `React.PureComponent` class are treated as pure components.

example:

```
import React from ‘react’;

export default class Test extends React.PureComponent{
   render(){
      return <h1>Welcome to TupleScale</h1>;
   }
}
```

output:

Welcome to TupleScale

EXAMPLE:

```
import React,{Component} from "react";
import './App.css'

class App extends Component{
   render() {
      return (
         <div className="App">
         <PurreComp />
         </div>
      )
   }
}
export default App


import React, {PureComponent} from 'react'
class PureComp extends PureComponent{
   render(){
      return (
         <div>
            PureComponent
         </div>
      )
   }
}
export default App
```

output

PureComponent

controlled and uncontrolled components

contol component:

Element data can be contolled by parent component through callbacks like onChange()

```
import React from "react";
import Test from './Test';

const Controlled =() =>{
   const [name,setName] =React.useState('');
   const changeHandler = e => {
      setName(e.target.value);

   }
   return (
      <div>
         <center>
            Name:{name}<br />
            <input type='text' onChange={changeHandler}/> <br />
```

```
                <Test changeHandler={changeHandler} />
            </center>
        </div>
    )
}
export default Controlled
```

```
import React from "react";

const Test =(props)=>{
    return(
        <div>
            <center>
                <input type="text" onChange={props.changeHandler}/>
            </center>
        </div>
    )
}
export default Test
```

uncontolled components

Elements data can be contolled by the DOM itself

```
import React from "react";

function Example(){
    const inputRef=React.useRef('');
    const submitHandler = e =>{
        e.preventDEfault();
        alert(inputRef.current.value)
    }
    return(
        <div>
        <form onSubmit={submitHandler}>
        <input type="text" ref={inputRef}/>
        <input type="submit"value="submit"/>
        </form>
        </div>
    )
}
export default Uncontrolled
```