## 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,

**COMPONENT NAMING:**

The convention:

[Domain]|[page/context]|[componentName]|[type]

examples:

Domain,page/context,component,type

### Component types

For now, we identified only 5 types of components:

- •View component: only display a rendering of data (no API calls or internal actions)
- •Button component: only display an actionable view
- •Connect component: legacy connect components
- •forms components:Input,Upload, etc …
- •in case of HoC components, we add Component to the wrapped component, and the HoC take the original component name

### Using Camel Case for File Names

Camel case is the practice of writing phrases without spaces or punctuation, indicating the separation of words with a single capitalized letter.

```
const AppContainer = () => {
  return (
    <div className="App">
      <h1>I Have Camel Case Naming</h1>
      <p>And I'm proud of it!</p>
    </div>
  );
}

export default App
```

**Using Kebab Case for File Names**

Kebab case is a programming variable naming convention where a developer replaces the spaces between words with a dash.

```
app-container.jsx
const AppContainer = () => {
  return (
    <div className="App">
      <h1>I Have Kebab Case Naming</h1>
      <p>And I'm also proud of it!</p>
    </div>
  );
}

export default App
```

**Using Pascal Case for Both File and Component Name**

Pascal case is following the same format as the camel case. In addition, it requires the first letter to be uppercase as well, while the camel case does not.

```
AppContainer.jsx
const AppContainer = () => {
  return (
    <div className="App">
      <h1>I Pascal Case Naming</h1>
      <p>And I'm also proud of it!</p>
    </div>
  );
}

export default App
```

**Class Component**

A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a render() method, this method returns HTML.

**Example:**

**app.js**

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

import logo from './logo.svg';

import './App.css';

import Greet from './components/Greet'

import Welcome from './components/Welcome'

class App extends Component{

        render(){

                return(

                  <div className="App">

                   <Greet/>

                   <welcome />

                  </div>

                );

        }

}

export default App;
```

**Greet.js**

```
import React from "react";

const Greet=()=><h1>Hello Tuplescale</h1>

export default Greet
```

**Welcome.js**

```
import React {Component} from "react";
class Welcome extends Component{
render(){
        return <h1>Class Component</h1>
        }
}
export default Welcome
```

**output:**

```
         Hello Tuplescale
         Class Component
```

## Functional component

A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand

**Example**

**App.js**

```
import React,{Component} from "react";
import logo from './logo.svg';
import './App.css';
import Greet from './components/Greet'
class App extends Component{
        render(){
                return(
                  <div className="App">
                   <Greet/>
                  </div>
                );
        }
}
export default App;
```

**Greet.js**

```
import React from "react";

export const Greet=()=><h1>Hello Tuplescale</h1>
```

**output**

```
Hello Tuplescale
```