* A component describes what you want to see on the screen
* Components let you split the UI into independent, reusable pieces
* components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.
* Components are like JavaScript functions
* In React Components are defined in a simple JavaScript file.
* In React each component return as a custom html element.
* Components are re-usable
* Components are invoked one another
* Components are two Types
  + Functional Components
    - are just JavaScript functions
    - accept input as props and return html as UI
  + Class Components
    - must contain render method
    - class components are basically es6 classes
    - accept input as props and return html as UI
    - maintains state (private to that component)
    - provide life cycle hooks
    - are state full/smart/container

Difference between Function and Class components

|  |  |
| --- | --- |
| Function | Class |
| Simple Functions | More feature rich |
| Use function components as much as Possible | Maintain their own private data-state |
| Absence of this keyword | Complex UI Logic |
| Solution without using state | Provide life cycle hooks |
| Mainly responsible for UI |  |
| stateless/dumb/presentational | stateful/smart/container |
|  |  |

## Function and Class Components

The simplest way to define a component is to write a JavaScript function (Function Component):

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

## This function is a valid React component because it accepts a single “props” (which stands for properties) object argument with data and returns a React element. We call such components “function components” because they are literally JavaScript functions.

Class Component:

You can also use an [ES6 class](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes) to define a class component:

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

## Rendering a Component

function Welcome() {

return <h1>Hello, React</h1>;

}

ReactDOM.render(

<Welcome />;

document.getElementById('root')

);

**Note: Always start component names with a capital letter.**

React treats components starting with lowercase letters as DOM tags. For example, <div /> represents an HTML div tag, but <Welcome /> represents a component and requires Welcome to be in scope.

## Composing Components

Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail. A button, a form, a dialog, a screen: in React apps, all those are commonly expressed as components.

For example, we can create an App component that renders Welcome many times:

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

function App() {

return (

<div>

<Welcome name="Sara" /> <Welcome name="Cahal" /> <Welcome name="Edite" /> </div>

);

}

ReactDOM.render(

<App />,

document.getElementById('root')

);

## Extracting Components

In React components split into smaller components.

For example, consider this Comment component:

function Comment(props) {

return (

<div className="Comment">

<div className="UserInfo">

<img className="Avatar"

src={props.author.avatarUrl}

alt={props.author.name}

/>

<div className="UserInfo-name">

{props.author.name}

</div>

</div>

<div className="Comment-text">

{props.text}

</div>

<div className="Comment-date">

{formatDate(props.date)}

</div>

</div>

);

}

This component can be tricky to change because of all the nesting, and it is also hard to reuse individual parts of it. Let’s extract a few components from it.

First, we will extract Avatar:

function Avatar(props) {

return (

<img className="Avatar" src={props.user.avatarUrl} alt={props.user.name} /> );

}

We can now simplify Comment a tiny bit:

function Comment(props) {

return (

<div className="Comment">

<div className="UserInfo">

<Avatar user={props.author} /> <div className="UserInfo-name">

{props.author.name}

</div>

</div>

<div className="Comment-text">

{props.text}

</div>

<div className="Comment-date">

{formatDate(props.date)}

</div>

</div>

);

}

Next, we will extract a UserInfo component that renders an Avatar next to the user’s name:

function UserInfo(props) {

return (

<div className="UserInfo"> <Avatar user={props.user} /> <div className="UserInfo-name"> {props.user.name} </div> </div> );

}

This lets us simplify Comment even further:

function Comment(props) {

return (

<div className="Comment">

<UserInfo user={props.author} /> <div className="Comment-text">

{props.text}

</div>

<div className="Comment-date">

{formatDate(props.date)}

</div>

</div>

);

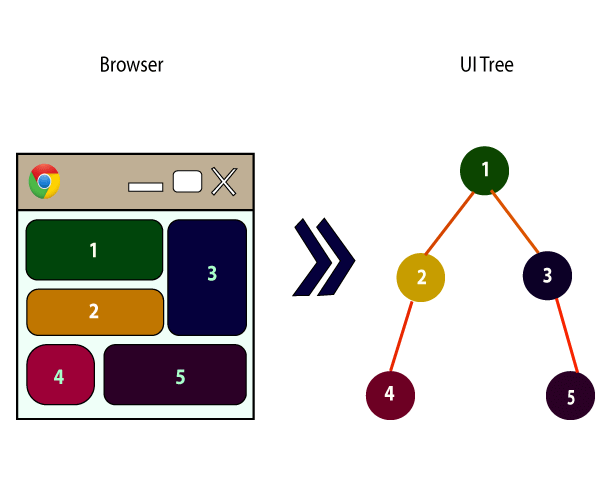
}

 a small logical group of code, which is known as components.

A Component is considered as the core building blocks of a React application.

It makes the task of building UIs much easier. Each component exists in the same space, but they work independently from one another and merge all in a parent component, which will be the final UI of your application.

Every React component have their own structure, methods as well as APIs. They can be reusable as per your need. For better understanding, consider the entire UI as a tree. Here, the root is the starting component, and each of the other pieces becomes branches, which are further divided into sub-branches.



In ReactJS, we have mainly two types of components. They are

1. Functional Components
2. Class Components

Functional Components

In React, function components are a way to write components that only contain a render method and don't have their own state. They are simply JavaScript functions that may or may not receive data as parameters. We can create a function that takes props(properties) as input and returns what should be rendered. A valid functional component can be shown in the below example.

1. function WelcomeMessage(props) {
2. **return** <h1>Welcome to the , {props.name}</h1>;
3. }

The functional component is also known as a stateless component because they do not hold or manage state.

Class Components

Class components are more complex than functional components. It requires you to extend from React. Component and create a render function which returns a React element. You can pass data from one class to other class components. You can create a class by defining a class that extends Component and has a render function. Valid class component is shown in the below example.

1. **class** MyComponent **extends** React.Component {
2. render() {
3. **return** (
4. <div>This is main component.</div>
5. );
6. }
7. }

The class component is also known as a stateful component because they can hold or manage local state.