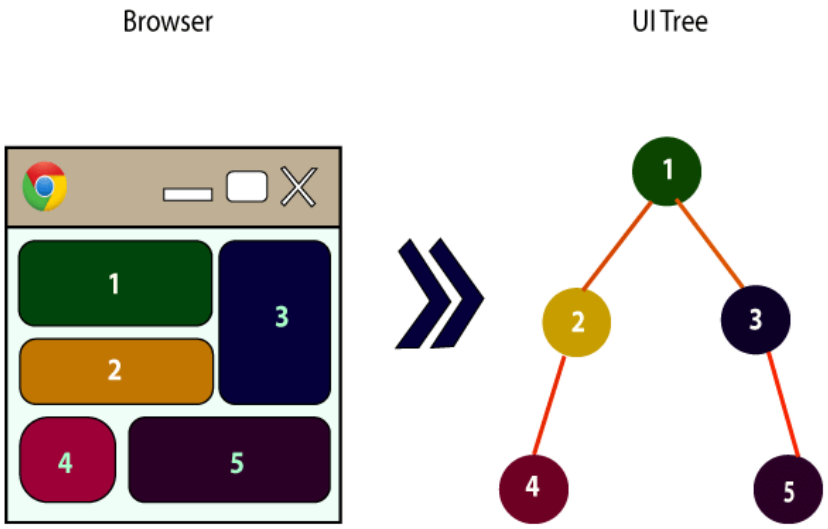
React Components

- A Component is considered as the core building blocks of a React application.
- Components are a fundamental concept in React, and they are the building blocks of a React application.
- A component is a reusable, self-contained piece of code that encapsulates a specific piece of functionality or a part of the user interface.
- React applications are typically composed of multiple components that work together to create the overall user interface.
- In React Components are defined in a simple JavaScript file.
- In React each component return as a custom html element.
- Components are re-usable and nested.

React Components

- Every React component have their own structure(UI), methods as well as API calls.
- They can be reusable as per your need.
- 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.



React Components

- Components are two Types
  - Functional Components
  - Class Components

Functional Components

- Functional Components are just like JavaScript functions
- Functional Components accept input as **props**(short for properties) and return html as react elements.
- Functional Components also known as stateless components or presentational components.
- Functional component is also known as a stateless component because they do not hold or manage state.
- Suitable for simple components that don't need to manage state or lifecycle methods.
- React Hooks, introduced in React 16.8, allow functional components to use state and other React features.

```
// Example of a functional component
const Greeting = (props) => {
  return <p>Hello, {props.name}!</p>;
};
```

Note: component's name MUST start with an upper case letter

React Props

- In React Props stand for "Properties"
- Props are arguments passed into react component
- They are read-only .
- React Props are like function arguments in JavaScript and attributes in HTML.
- It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.
- It gives a way to pass data from one component to other components.
- Props are passed to the component in the same way as arguments passed in a function.
- Props are immutable so we cannot modify the props from inside the component.
- Inside the components, we can add attributes called props. These attributes are available in the component as this. Props and can be used to render dynamic data in our render method.

Composing Components

- Components can refer to other components in their output.

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')
);
```

Class Components

- Class components are basically Defined as ES6 classes.
- Class components are extend '**React.Component**'
- Class components are more complex than functional components.
- Class components Can manage local state and have access to lifecycle methods.
- Class Components should create a render function to returns a React element.
- Class components are also known as a stateful components or container components because they can hold or manage local state.(private to that component)
- Prior to React 16.8, class components were the primary way to manage state and lifecycle methods.
- ***class MyComponent extends React.Component {***

  render() {

  return (

  *** This is main component. ***

  );

  }

  }

Class Components

```
// Example of a class component
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
```

Function and Class Components

- Function Component

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- 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')
);
```

React Components

```jsx
// Functional component
const Greeting = (props) => {
  return <p>Hello, {props.name}!</p>;
};

// Class component
class App extends React.Component {
  render() {
    return (
      <div>
        <Greeting name="John" />
        <Greeting name="Jane" />
      </div>
    );
  }
}

ReactDOM.render(<App />, document.getElementById('root'));
```

React State

- In React, the "state" is a JavaScript object that represents the current condition or data of a component
- State is a fundamental concept and plays a crucial role in building dynamic and interactive user interfaces.
- Each component in React can have its own state, which can be modified over time, leading to updates in the UI.
- The state is an updatable structure that is used to contain data or information about the component.
- The state in a component can change over time.
- A component with the state is known as stateful components.
- It is the heart of the react component which determines the behavior of the component and how it will render.
- The change in state over time can happen as a response to user action or system event.
- They are also responsible for making a component dynamic and interactive.
- A state represents the component's state or information.

- Declaring State:
- State is typically declared in a class component's constructor using the **this.state** syntax. It is initialized with an object representing the initial state.

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
    };
  }

  // Other component methods...
}
```

- Updating State::
- The **setState** method is used to update the state. It takes an object that represents the partial state to be updated. React then merges this object with the current state.

```
increment = () => {
  this.setState({ count: this.state.count + 1 });
};
```

- **Immutable State:**
- State should be treated as immutable in React. Instead of modifying the state directly, you provide a new object that represents the updated state. This ensures that React can correctly determine when to re-render components.

```
// Incorrect way (mutable state)
this.state.count = this.state.count + 1;

// Correct way (immutable state)
this.setState({ count: this.state.count + 1 });
```

- **Functional State Updates:**
- When the new state is computed based on the previous state, you can use a functional update to ensure that the update is based on the most recent state.

```
increment = () => {
  this.setState((prevState) => ({
    count: prevState.count + 1,
  }));
};
```

- **Passing State as Props:**
- State can be passed down to child components as props, allowing child components to access and display the data.

```
<ChildComponent count={this.state.count} />
```

- **Initializing State with Props:**
- You can initialize a component's state based on the props it receives using the constructor.

```
constructor(props) {
  super(props);
  this.state = {
    count: props.initialCount,
  };
}
```

- **Local vs. Global State:**
- State is local to the component that declares it. For managing state across multiple components, you might use state management solutions like React Context or external state management libraries like Redux.

```
// Local state in a component
class ComponentA extends React.Component {
  state = {
    data: 'Local State',
  };
}
```

React State Vs Props

SN	Props	State
1.	Props are read-only.	State changes can be asynchronous.
2.	Props are immutable.	State is mutable.
3.	Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
4.	Props can be accessed by the child component.	State cannot be accessed by child components.
5.	Props are used to communicate between components.	States can be used for rendering dynamic changes with the component.
6.	Stateless component can have Props.	Stateless components cannot have State.
7.	Props make components reusable.	State cannot make components reusable.
8.	Props are external and controlled by whatever renders the component.	The State is internal and controlled by the React Component itself.