# States In React.Js

The **state** is an **instance** of React **Component Class** can be defined as an object of a set of observable properties that control the behavior of the component. In other words, the State of a component is an object that **holds some information that may change over the lifetime of the component**. For example, let us think of the clock that we created in this article, we were calling the **render**() method every second **explicitly**, but React provides a better way to achieve the same result and that is by using State, storing the value of time as a member of the component's state.

We will look into this more elaborately later in this note.

## **Difference** of Props and State

We have already learned about Props and we got to know that Props are also objects that hold **information** to control the behavior of that particular **component**, sounds familiar to State indeed but **props** and **states** are nowhere near be same. Let us differentiate the two.

- Props are **immutable** i.e. once set the props cannot be changed, while State is an **observable** object that is to be used to hold data that may change over time and to control the behavior after each change.
- **States** can be used in **Class Components**, Functional components with the use of React **Hooks** (**useState** and other methods) while Props don't have this limitation.

- While **Props** are set by the **parent component**, **State** is generally updated by event handlers. For example, let us consider the toggle the theme of the **Code Studio(Coding Ninjas)** page. It can be implemented using State where the probable values of the State can be either light or dark and upon selection, the **IDE changes its color**.

## Conventions of Using State in React:

- State of a component should prevail throughout the **lifetime**, thus we must first have some initial state, to do so we should define the State in the constructor of the component's class. To define a state of any Class we can use the sample format below.

```
Class MyClass extends React.Component
{
        constructor(props)
        {
                super(props);
                this.state = { attribute : "value" };
        }
}
```

- State should **never** be **updated explicitly**. React uses an **observable** object as the state that observes what changes are made to the state and helps the component behave accordingly. For example, if we update the state of any **component** like the following the webpage will not **re-render** itself because **React State** will not be able to detect the changes made.

*this.state.attribute = "new-value";*

- Thus, React provides its own method **setState**(). **setState**() method takes a **single parameter** and expects an object which should contain the set of values to be updated. Once the update is done the method implicitly calls the **render**() method to repaint the page. Hence, the correct method of updating the value of a state will be similar to the code below.

  this.setState({attribute: "new-value"});

- The only time we are allowed to define the state explicitly is in the constructor to provide the initial state.
- React is highly efficient and thus uses **asynchronous** state updates i.e. React may update multiple **setState**() updates in a single go. Thus using the value of the current state may not always generate the desired result. For example, let us take a case where we must keep a count (**Likes** of a **Post**). Many developers may miswrite the code as below.

  *this.setState({counter: this.state.count + this.props.diff});*

- Now due to **asynchronous** processing, **this**.**state**.**count** may produce an **undesirable** result. A more appropriate approach would be to use the following.

  *this.setState((prevState, props) => ({*
  *counter: prevState.count + props.diff*
  *}));*

- State updates are **independent**. The state object of a component may contain **multiple attributes** and React allows to use **setState**() function to update only a subset of those attributes as well as using multiple setState() methods to update each attribute value independently. For example, let us take the following component state into account.

  *this.state = {*
  *darkTheme: False,*
  *searchTerm: ''*
  *};*

  The above definition has two attributes we can use a single **setState**() method to update both together, or we can use separate **setState**() methods to update the attributes **independently**. **React** internally **merges** setState() methods or updates only those attributes which are needed.

  After going through the article you should have a clear concept of **State** in React, but other than the constructor and **render** methods can we add **user-defined** functions as well? Yes, we can also create user-defined functions inside a class but **how to call them?** React provides a few special methods that are called at some proper context that solves this **problem**.