Q. React :-

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called "components".

Q. 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.

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

Create a Class component called Car

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

Function Component

Here is the same example as above, but created using a Function component instead.

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

Create a Function component called Car

```
function Car() {

  return <h2>Hi, I am a Car!</h2>;

}
```

Q. State :-

The state is a built-in React object that is used to contain data or information about the component. A component’s state can change over time; whenever it changes, the component re-renders. The change in state can happen as a response to user action or system-generated events and these changes determine the behavior of the component and how it will render.

```
class Greetings extends React.Component {

  state = {

    name: "World"

  };

  updateName() {

    this.setState({ name: "Simplilearn" });

  }

  render() {

      return(

          <div>

              {this.state.name}

          </div>

      )

  }
}
```

- A state can be modified based on user action or network changes
- Every time the state of an object changes, React re-renders the component to the browser
- The state object is initialized in the constructor
- The state object can store multiple properties
- this.setState() is used to change the value of the state object
- setState() function performs a shallow merge between the new and the previous state

The setState() Method

State can be updated in response to event handlers, server responses, or prop changes. This is done using the setState() method. The setState() method enqueues all of the updates made to the component state and instructs React to re-render the component and its children with the updated state.

Always use the setState() method to change the state object, since it will ensure that the component knows it's been updated and calls the render() method.

Now that we are familiar with the concept of a state in React, let's have a look at how it is implemented in a React web application.

```
class Bike extends React.Component {

  constructor(props) {

    super(props);
    this.state = {
     make: "Yamaha",
     model: "R15",
     color: "blue"
   };
 }
 changeBikeColor = () => {
   this.setState({color: "black"});
 }
 render() {
   return (
     <div>
       <h2>My {this.state.make}</h2>
       <p>
         It is a {this.state.color}
         {this.state.model}.
       </p>
       <button
         type="button"
```

```
          onClick={this.changeBikeColor}
        >Change color</button>
      </div>
    );
  }
}
```

Q. Props :-

Props are arguments passed into React components.

Props are passed to components via HTML attributes.

React Props are like function arguments in JavaScript *and* attributes in HTML.

To send props into a component, use the same syntax as HTML attributes:

Example

Add a "brand" attribute to the Car element:

```
const myElement = <Car brand="Ford" />;
```

The component receives the argument as a props object:

Example

Use the brand attribute in the component:

```
function Car(props) {

  return <h2>I am a { props.brand }!</h2>;

}
```

Pass Data

Props are also how you pass data from one component to another, as parameters.

Example

Send the "brand" property from the Garage component to the Car component:

```
function Car(props) {

  return <h2>I am a { props.brand }!</h2>;

}
```

```
function Garage() {

  return (

    <>

      <h1>Who lives in my garage?</h1>

      <Car brand="Ford" />

    </>

  );

}


const root =
ReactDOM.createRoot(document.getElementById('root'));

root.render(<Garage />);
```

If you have a variable to send, and not a string as in the example above, you just put the variable name inside curly brackets:

Example

Create a variable named carName and send it to the Car component:

```
function Car(props) {

  return <h2>I am a { props.brand }!</h2>;

}


function Garage() {

  const carName = "Ford";

  return (

    <>

      <h1>Who lives in my garage?</h1>

      <Car brand={ carName } />
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garage />);
```

Or if it was an object:

Example

Create an object named carInfo and send it to the Car component:

```
function Car(props) {
  return <h2>I am a { props.brand.model }!</h2>;
}

function Garage() {
  const carInfo = { name: "Ford", model: "Mustang" };
  return (
    <>
      <h1>Who lives in my garage?</h1>
      <Car brand={ carInfo } />
    </>
  );
}
```

```
const root =
ReactDOM.createRoot(document.getElementById('root'));

root.render(<Garage />);
```