REACT ASSIGNMENT : 19/09/2022 (SAKSHAM DANI)

1. **What is ReactJS ?**

   ReactJS is a JavaScript library used for building reusable UI components which present data that changes over time.
   React features :
   - JSX : JavaScript syntax extension
   - Components : reusable
   - Unidirectional data flow and Flux
   - Virtual DOM: increases performance (faster than real DOM)

2. **Rendering elements in DOM.**

   In order to render any element into the Browser DOM, we need to have a container or root DOM element. Applications built with just React usually have a single root DOM node.To render a React element, first pass the DOM element to `ReactDOM.createRoot()` , then pass the React element to `root.render()`:

```
const root = ReactDOM.createRoot(
  document.getElementById('root')
);
const element = <h1>Hello, world</h1>;
root.render(element);
```

   Output : Hello, world on the screen

   OR we can make a separate file for eg. App.js and write the following :

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

class App extends Component {

  render() {
    return (
       <div>
         <h1>Hello, world</h1>
       </div>

    );
  }
}
export default App;
```

   Both of them will have same output on the screen.

3. **How does react render and Update the UI?**

   React elements are immutable. Once you create an element, you can't change its children or attributes. A way to update the UI is to create a new element, and pass it to **root.render()**

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

function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  root.render(element);
}

setInterval(tick, 1000);
```

It calls **root.render()** every second from a **setInterval()** callback.

- In practice, most React apps only call **root.render() once** and use **stateful components** to manage state of the elements.
- **React Only Updates What's Necessary**
  React DOM compares the element and its children to the previous one, and only applies the DOM updates necessary to bring the DOM to the desired state.
  Even though we create an element describing the whole UI tree on every tick, only the text node whose contents have changed gets updated by React DOM.

4. What are React Components?

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.
There are 2 ways to define a component : function and class based components.

5. Function based and Class Based Components

Function based : Functional components are simply javascript functions. We can create a functional component in React by writing a javascript function. These functions may or may not receive data as parameters.

```
const Democomponent=()=>
{
    return <h1>Welcome Message!</h1>;
}
```

Class based : The class components are a little more complex than the functional components. The functional components are not aware of the other components in your program whereas the class components can work with each other. We can pass data from one class component to other class components.

```
class Democomponent extends React.Component

{
```

```
render(){
            return <h1>Welcome Message!</h1>;
    }
}
```

Both the Above components are equivalent.

6. **What are props?**
   React allows us to pass information to a Component using something called props (stands for properties). Props are objects which can be used inside a component.

   We can pass props to any component as we declare attributes for any HTML tag. Have a look at the below code snippet:

   <DemoComponent sampleProp = "HelloProp" />

   - We can pass as many props as we want to a component.
   - whatever information is carried to a component using props is stored inside an object.
   - Props are read-only. We are not allowed to modify the content of a prop

7. **What is 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.
   - These changes determine the behavior of the component and how it will render.
   - 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

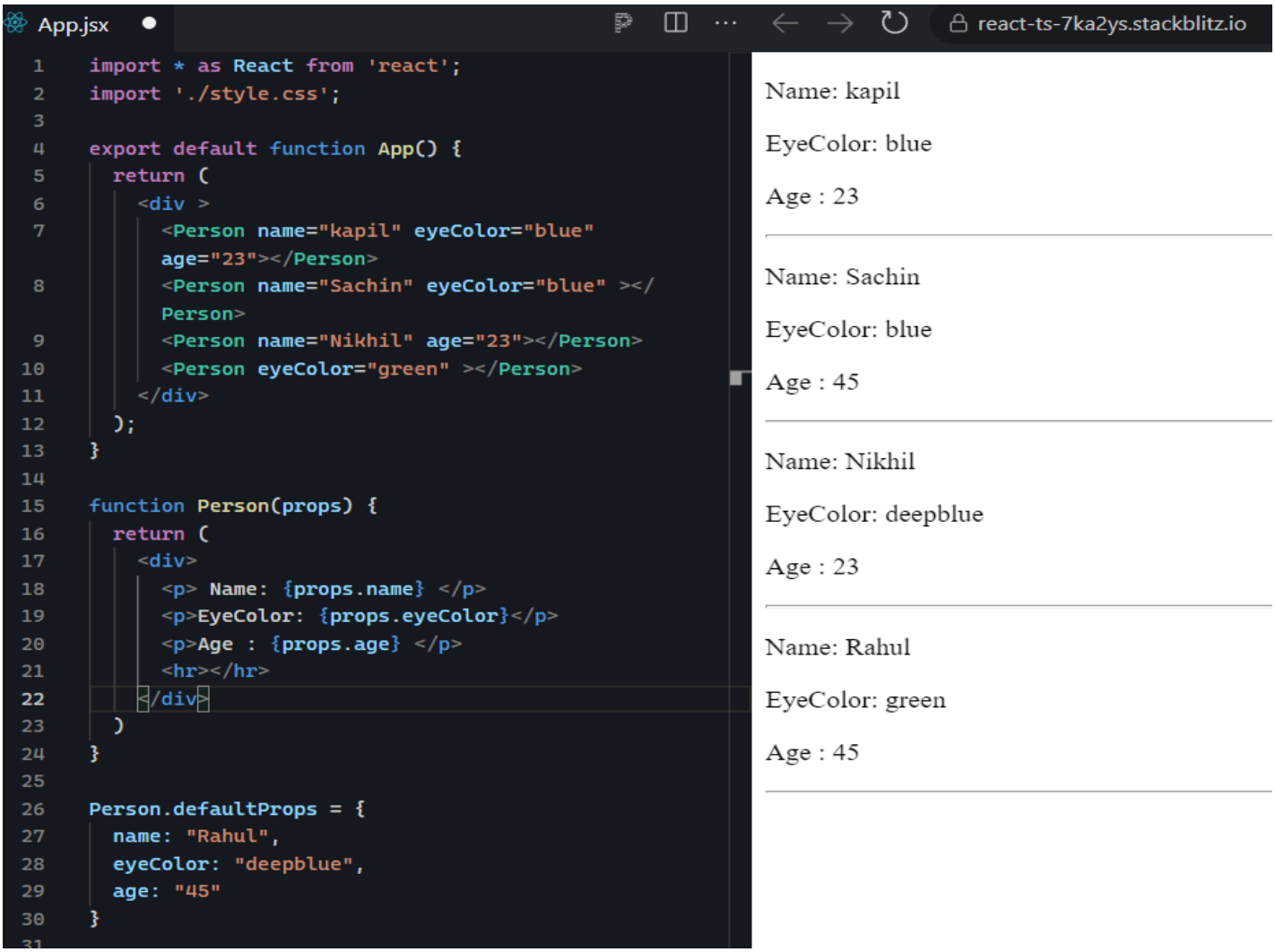
8. **Props vs State.**

	State	Props
Use Case	State is used to store the data of the components that have to be rendered to the view	Props are used to pass data and event handlers to the children components
Mutability	State holds the data and can change over time	Props are immutable—once set, props cannot be changed

Updation	Event handlers generally update state	The parent component sets props for the children components

9. **Default props and type checking using proptypes.**

Default props : The defaultProps is a React component property that allows you to set default values for the props argument. If the prop property is passed, it will be changed. The defaultProps can be defined as a property on the component class itself, to set the default props for the class.



Type checking proptypes

We can use PropTypes for validating any data we are receiving from props. This is called Type Checking with Prop Types.
As JavaScript is dynamically typed the data type of variables are assigned at runtime as react is also written in JavaScript there is no way to check prop-types before rendering that is why react provides prop types to help us validate the data being received from props.

Type-checking will warn us with a console message if we pass a type different from the data type specified by propTypes.

```
import PropTypes from 'prop-types';
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css'

// Component
class ClassComponent extends React.Component{
    render(){
        return(
          <div>
           {/* printing all props */}
           <h1>
           {this.props.goodsPrice}
           </h1>
          </div>
       );
   }
}

// Validating prop types
ClassComponent.propTypes = {
  goodsPrice: PropTypes.number
}

// Creating default props
ClassComponent.defaultProps = {
  goodsPrice: "GeeksForGeeks"
}

ReactDOM.render(
    <ClassComponent  />,
    document.getElementById("root")
);
```

10. Handling events in react

Handling events with React elements is very similar to handling events on DOM elements. There are some syntax differences:

- React events are named using camelCase, rather than lowercase.
- With JSX you pass a function as the event handler, rather than a string.

For example, the HTML:

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

is slightly different in React:

```
<button onClick={activateLasers}>
        Activate Lasers
</button>
```

- The difference is that you cannot return false to prevent default behavior in React. You must call preventDefault explicitly
- To pass an argument to an event handler, use an arrow function.

```
function Football() {
  const shoot = (a) => {
    alert(a);
  }

  return (
    <button onClick={() => shoot("Goal!")}>Take the shot!</button>
  );
}
```

11. Life cycle methods of class components

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases. The three phases are: **Mounting, Updating,** and **Unmounting.**

Mounting
Mounting means putting elements into the DOM.
React has four built-in methods that gets called, in this order, when mounting a component:

1. `constructor()`
2. `getDerivedStateFromProps()`
3. `render()`
4. `componentDidMount()`

The **render()** method is required and will always be called, the others are optional and will be called if you define them.

- **`componentDidMount()`**
  The componentDidMount() method is called after the component is rendered.
  This is where you run statements that requires that the component is already placed in the DOM

Updating

The next phase in the lifecycle is when a component is updated.

A component is updated whenever there is a change in the component's state or props.
React has five built-in methods that gets called, in this order, when a component is updated:

1. `getDerivedStateFromProps()`
2. `shouldComponentUpdate()`
3. `render()`
4. `getSnapshotBeforeUpdate()`
5. `componentDidUpdate()`

The render() method is required and will always be called, the others are optional and will be called if you define them.

- **`shouldComponentUpdate()`**

  In the shouldComponentUpdate() method you can return a Boolean value that specifies whether React should continue with the rendering or not.

  The default value is true.

T

- **`componentDidUpdate()`**

  The componentDidUpdate method is called after the component is updated in the DOM.

Unmounting

The next phase in the life cycle is when a component is removed from the DOM, or unmounting as React likes to call it.

React has only one built-in method that gets called when a component is unmounted:

- componentWillUnmount()

- **`componentWillUnmount()`**

  The componentWillUnmount method is called when the component is about to be removed from the DOM.