**REACT**

* **What is React?**

1. Open source library for building user interfaces
2. Not a framework
3. Used to build single page application
4. Allows us to create reusable components
5. Focus on UI
6. Has rich ecosystems plays well with other lib.
7. More than capable of building full fledged app
8. Current version of React.JS is V18.0.0 (April 2022).
9. React.JS was first used in 2011 for Facebook's Newsfeed feature.
10. Facebook Software Engineer, Jordan Walke, created it.

* **Why learn react?**

1. Markup reusability
2. State Management
3. Created and maintained by facebook
4. More than 100k starts on Github
5. Huge community support
6. In demand skillset

* **How react works?**

1. Component based architecture
   1. App component
   2. Header component
   3. Footer component
   4. Nav component
   5. Main component
2. Creates a VIRTUAL DOM in memory
   1. Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.
3. React only changes what needs to be changed!
4. React is declarative
   1. Tell react what you want and react will build the actual UI

* **Prerequistites**

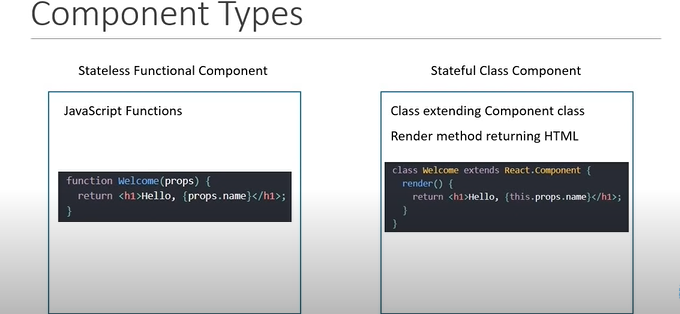
1. HTML, CSS and
2. JavaScript fundamentals like ‘this’ , filter, map, reduce
3. ES6 feature like
   1. Let & const, arrow functions, template literals, default parameters, object literals, rest and spread operators and destructuring assignment.

* **Roadmap**

1. Fundamentals - HTTP - Routing - Redux - Utilities

* **Components**

1. Describes a part of user interface
2. They are re-usable and can be nested inside other components
3. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.
4. Types:
   1. Stateless functional component
   2. Stateful class component



* **ES6 features**

1. ECMAScript 6
2. ECMAScript was created to standardize JavaScript and ES6 is the 6th version of ECMAScript and also know as ECMAScript 2015
3. Concepts to know:
   1. Classes **[link](https://www.w3schools.com/react/react_es6_classes.asp)**
   2. Arrow Functions **[link](https://www.w3schools.com/react/react_es6_arrow.asp)**
   3. Variables(let, const, var) **[link](https://www.w3schools.com/react/react_es6_variables.asp)**
   4. Array Methods like map() **[link](https://www.w3schools.com/react/react_es6_array_methods.asp)**
   5. Destructuring  **[link](https://www.w3schools.com/react/react_es6_destructuring.asp)**
   6. Modules **[link](https://www.w3schools.com/react/react_es6_modules.asp)**
   7. Ternary Operator **[link](https://www.w3schools.com/react/react_es6_ternary.asp)**
   8. Spread Operator **[link](https://www.w3schools.com/react/react_es6_spread.asp)**

* **React JSX**
  + **[JSX intro](https://www.w3schools.com/react/react_jsx.asp)**

1. JSX stands for JavaScript XML.
2. JSX allows us to write HTML in React.
3. JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods.
4. JSX converts HTML tags into react elements.
5. JSX allows us to write HTML directly within the JavaScript code.
6. *“You are not required to use JSX, but JSX makes it easier to write React applications*.”

* **Without JSX**

1. Rendering HTML in React without JSX

|  |
| --- |
| const myElement = React.createElement('h1', {}, 'I do not use JSX!');  const root = ReactDOM.createRoot(document.getElementById('root'));  root.render(myElement); |

* **Using JSX**

|  |
| --- |
| const myElement = <h1>I Love JSX!</h1>;  const root = ReactDOM.createRoot(document.getElementById('root'));  root.render(myElement); |

* **Expressions in JSX**

1. With JSX you can write expressions inside curly braces { }.
2. The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result:

|  |
| --- |
| const myElement = <h1>React is {5 + 5} times better with JSX</h1>; |

* **One top level element**

1. JSX will return only one HTML element
2. The HTML code must be wrapped in ONE top level element.
3. So if you like to write two paragraphs, you must put them inside a parent element, like a div element or you can use fragment (empty element < > < />)

|  |  |
| --- | --- |
| const myElement = (  <div>  <p>I am a paragraph.</p>  <p>I am a paragraph too.</p>  </div>); | const myElement = (  <>  <p>I am a paragraph.</p>  <p>I am a paragraph too.</p>  </>); |

* **Errors in JSX**

1. JSX will throw an error if the HTML is not correct, or if the HTML misses a parent element. ( returning two elements without fragment or div)
2. JSX follows XML rules, and therefore HTML elements must be properly closed.

* **Attribute class = className**

1. The class attribute is a much used attribute in HTML, but since JSX is rendered as JavaScript, and the class keyword is a reserved word in JavaScript, you are not allowed to use it in JSX

|  |
| --- |
| const myElement = <h1 className="myclass">Hello World</h1>; |

* **Rendering JS inside JSX**

1. Note that in order to embed a JavaScript expression inside JSX, the JavaScript must be wrapped with curly braces, {}.

|  |
| --- |
| const x = 5;  const myElement = <h1>{(x) < 10 ? "Hello" : "Goodbye"}</h1>; |

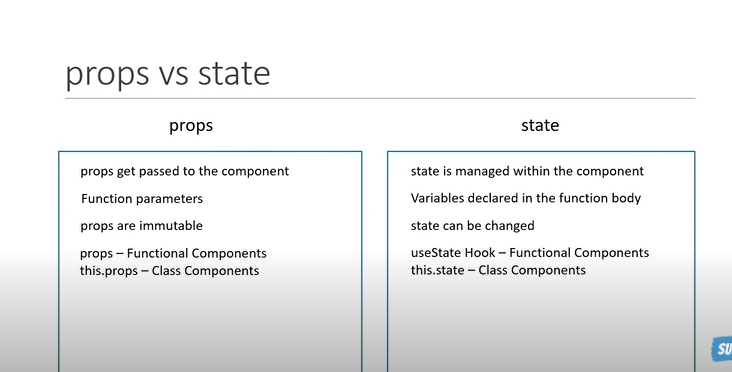
* **Props**

1. *“props”* stands for properties
2. React Props are like function arguments in JavaScript and attributes in HTML.
3. React Props are read-only! You will get an error if you try to change their value.
4. [Props an overview](https://www.w3schools.com/react/react_props.asp)

|  |
| --- |
| const myElement = <Car brand="Ford" />; |

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

|  |
| --- |
| 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 />); |



* **States**

1. In React.js, a state is a built-in feature that allows you to store and manage data within a component.
2. States are typically used for dynamic content, such as user input, data fetching, or any other aspect of your application that needs to be updated and reflected in the user interface.

|  |
| --- |
| In Class Component:  import React, { Component } from 'react';  class MyComponent extends Component {  constructor(props) {  super(props);  this.state = { count: 0 };  }  render() {  return (  <div>  <p>Count: {this.state.count}</p>  <button onClick={() => this.setState({ count: this.state.count + 1 })}>Increment</button>  </div>  );  }  } |

|  |
| --- |
| In Functional Component:  import React, { useState } from 'react';  function MyComponent() {  const [count, setCount] = useState(0);  return (  <div>  <p>Count: {count}</p>  <button onClick={() => setCount(count + 1)}>Increment</button>  </div>  );  } |

* **Destructuring the Props | States**

1. Props and states can destructured like normal javascript object

* **React Event Handlers**

1. Just like HTML DOM events, React can perform actions based on user events.
2. React has the same events as HTML: click, change, mouseover etc.
3. Adding Events:
   1. React events are written in camelCase syntax:
   2. onClick instead of onclick.
4. Event handlers:
   1. React event handlers are written inside curly braces:
   2. onClick={shoot} instead of onClick="shoot()".

|  |  |
| --- | --- |
| **REACT**  <button onClick={shoot}>Take the Shot!</button> | HTML  <button onclick="shoot()">Take the Shot!</button> |

* **React Hooks**

1. [What? Why? about Hooks](https://www.youtube.com/watch?v=cF2lQ_gZeA8&list=PLC3y8-rFHvwgg3vaYJgHGnModB54rxOk3&index=44)
2. [w3 schools precise intro](https://www.w3schools.com/react/react_hooks.asp)
3. **Prerequesites**
   1. Class component
   2. Functional component
   3. Props
   4. State
4. **What are Hooks?**
   1. Hooks were added to React in version 16.8.
   2. Hooks allow function components to have access to state and other React features without having to write a class.
   3. Hooks allow us to "hook" into React features such as state and lifecycle methods.
   4. State generally refers to application data or properties that need to be tracked i.e Student entity data needs to updated every.exams
5. **Three rules about Hooks**
   1. Hooks can only be called inside React functional component
   2. Hooks can only be called at top level of a component
   3. Hooks cannot be conditional
6. **Why Hooks?**
7. Eliminates the confusion of *‘this’* keyword which behaves typically in javascript
8. Easy to test the code as it has only funcitonal components
9. Simplified State Management with help of useState() hook
10. Complex state logic in functional components
11. Allow you to resuse stateful logic
12. [Importance of Hooks in React](D:\\Upskilling\\React\\SmartNotes)

* **UseState( )**

1. Accepts one ***object*** and Returns an ***array of object*** of length two
2. Const [currentValue, MethodToSetState] = useState(initialValue)
   1. Initial Value => initial value of the state property. Can be any types stirng, number, Date, user-defined objects like Customer, employee etc.
   2. currentValue => contains the current value of the object after the state changed
   3. MethodToSetState => contains a method which changes the state value as per definition
3. when you update the state using a state setter function, it's important to pass a function that calculates the new state based on the previous state, rather than passing a direct block of code that modifies the state directly.

|  |
| --- |
| MethodToSetState( prevState => prevState + 1) |

1. By passing a function to `setState`, you ensure that you are always working with the latest state.
2. React will call your function with the current state as an argument, and you can return the updated state based on that.
3. **Summary**
   1. The useState hook lets you add state to functional components.
   2. In classes, the state is always an object.
   3. With the useState hook, the state doesn’t have to be an object.
   4. The useState hook returns an array with elements
   5. The first element is the current value of the state, and the second element is a state setter function.
   6. New state value depends on the previous state value? You can pass a function to the setter function.
   7. When dealing with objects or arrays, always make sure to spread your state variable and then call the state setter function.

* **Component Life Cycle in React**

1. In react, component life cycle refers to the series of events and methods that a component goes through from its creation to its removal from DOM.
2. Class Component Lifecycle (with some commonly used methods):
   1. **Mounting Phase:**
      1. - `constructor()`: Initializes the component's state and binds event handlers. It is called before the component is mounted.
      2. - `static getDerivedStateFromProps(props, state)`: Rarely used, this method allows you to update the state based on changes in props before rendering.
      3. - `render()`: Renders the component and returns a React element to be displayed.
      4. - `componentDidMount()`: Invoked immediately after the component is inserted into the DOM. It's commonly used for data fetching, subscriptions, and initial setup.
   2. **Updating Phase:**
      1. `static getDerivedStateFromProps(props, state)`: Also used in the updating phase when props change.
      2. - `shouldComponentUpdate(nextProps, nextState)`: Allows you to control whether the component should re-render based on changes in props or state.
      3. - `render()`: Re-renders the component if necessary.
      4. - `getSnapshotBeforeUpdate(prevProps, prevState)`: Captures some information from the DOM before it changes, often used with scroll positions or other user interface-related data.
      5. - `componentDidUpdate(prevProps, prevState, snapshot)`: Invoked after the component's updates are flushed to the DOM. It's used for side effects, such as data fetching.
   3. **Unmounting Phase:**
      1. - `componentWillUnmount()`: Called just before the component is removed from the DOM. Used for cleaning up resources like subscriptions or timers.
3. **Functional Component Lifecycle**
   1. Functional components with React Hooks have a different approach for managing component lifecycle:
   2. **`useEffect()`:**
      1. Allows you to perform side effects in functional components. It replaces most of the lifecycle methods.
      2. It combines the behavior of `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` depending on the dependencies specified in the second argument (dependency array) or its absence.
      3. - You can use multiple `useEffect` calls in a single component to manage different side effects separately.