**https://www.javatpoint.com/react-hooks**

**What Is JSX:**

JSX is an Extension of JavaScript. JSX stands for JavaScript XML. JSX allows us to write HTML in React

**ReactJS Reconciliation:**

## **React 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, in this tutorial we will concentrate on 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.

### **Function Component**

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

## **React Props**

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:

# React State

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

# React Props Validation

Props are an important mechanism for passing the **read-only** attributes to React components. The props are usually required to use correctly in the component. If it is not used correctly, the components may not behave as expected. Hence, it is required to use **props validation** in improving react components.

**App.propTypes**

|  |  |
| --- | --- |
| PropTypes.any | The props can be of any data type. |
| **2.** | PropTypes.array | The props should be an array. |
| **3.** | PropTypes.bool | The props should be a boolean. |
| **4.** | PropTypes.func | The props should be a function. |
| **5.** | PropTypes.number | The props should be a number. |
| **6.** | PropTypes.object | The props should be an object. |
| **7.** | PropTypes.string | The props should be a string. |
| **8.** | PropTypes.symbol | The props should be a symbol. |

# React Component Life-Cycle

1. nitial Phase
2. Mounting Phase
3. Updating Phase
4. Unmounting Phase
5. <https://www.javatpoint.com/react-component-life-cycle>

# React Controlled Vs. Uncontrolled Component

## **Controlled Component**

A controlled component is bound to a value, and its changes will be handled in code by using **event-based callbacks**. Here, the input form element is handled by the react itself rather than the DOM

Controlled components have functions that govern the data passing into them on every **onChange** event occurs. This data is then saved to state and updated with setState() metho

## **Uncontrolled Component**

It is similar to the traditional HTML form inputs. Here, the form data is handled by the DOM itself. It maintains their own state and will be updated when the input value changes. To write an uncontrolled component, there is no need to write an event handler for every state update, and you can use a ref to access the value of the form from the DOM.

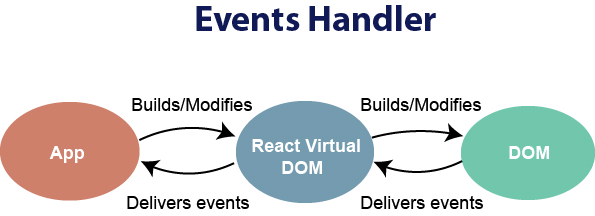
## **Difference table between controlled and uncontrolled component**

|  |  |  |
| --- | --- | --- |
| **SN** | **Controlled** | **Uncontrolled** |
| **1.** | It does not maintain its internal state. | It maintains its internal states. |
| **2.** | Here, data is controlled by the parent component. | Here, data is controlled by the DOM itself. |
| **3.** | It accepts its current value as a prop. | It uses a ref for their current values. |
| **4.** | It allows validation control. | It does not allow validation control. |
| **5.** | It has better control over the form elements and data. | It has limited control over the form elements and data. |

# React Events

An event is an action that could be triggered as a result of the user action or system generated event. For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.

React has its own event handling system which is very similar to handling events on DOM elements. The react event handling system is known as Synthetic Events. The synthetic event is a cross-browser wrapper of the browser's native event.



# React Keys

A key is a unique identifier. In React, it is used to identify which items have changed, updated, or deleted from the Lists

# React Refs

Refs is the shorthand used for **references** in React. It is similar to **keys** in React. It is an attribute which makes it possible to store a reference to particular DOM nodes or React elements

<https://www.javatpoint.com/react-refs>

# React Fragments

his render method can return **single** elements or **multiple** elements

<React.Fragment>  </React.Fragment>

<>   </>

## **React Router Installation**

React contains three different packages for routing. These are:

1. **react-router:** It provides the core routing components and functions for the React Router applications.
2. **react-router-native:** It is used for mobile applications.
3. **react-router-dom:** It is used for web applications design.

## **Components in React Router**

There are two types of router components:

* **<BrowserRouter>:** It is used for handling the dynamic URL.
* **<HashRouter>:** It is used for handling the static request.

# React Higher-Order Components

# React Code Splitting

The React app bundled their files using tools like **Webpack** or **Browserfy**. Bundling is a process which takes multiple files and merges them into a single file, which is called a **bundle**. The bundle is responsible for loading an entire app at once on the webpage. We can understand it from the below example.

1. **import** { add } from './math.js';
2. console.log(add(16, 26)); // 42

# React Context

Context allows passing data through the component tree without passing props down manually at every level.

In React application, we passed data in a top-down approach via props. Sometimes it is inconvenient for certain types of props that are required by many components in the React application. Context provides a way to pass values between components without explicitly passing a prop through every level of the component tree.

# React Hooks

Hooks are a new addition in React 16. They let you use React features such as state and lifecycle methods. And that allows in functional components

There are 3 rules for hooks:

* Hooks can only be called inside React function components.
* Hooks can only be called at the top level of a component.
* Hooks cannot be conditional.
* [Basic Hooks](https://legacy.reactjs.org/docs/hooks-reference.html#basic-hooks)
  + [useState](https://legacy.reactjs.org/docs/hooks-reference.html#usestate) : The React useState Hook allows us to track state in a function component.
  + [useEffect](https://legacy.reactjs.org/docs/hooks-reference.html#useeffect) : he useEffect Hook allows you to perform side effects in your components.

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

useEffect accepts two arguments. The second argument is optional.

useEffect(<function>, <dependency>)

* + [useContext](https://legacy.reactjs.org/docs/hooks-reference.html#usecontext) : React Context is a way to manage state globally.
* [Additional Hooks](https://legacy.reactjs.org/docs/hooks-reference.html#additional-hooks)
  + [useReducer](https://legacy.reactjs.org/docs/hooks-reference.html#usereducer) : The useReducer Hook is similar to the useState Hook.

It allows for custom state logic

It accept useReducer(<reducer>, <initialState>)

* + [useCallback](https://legacy.reactjs.org/docs/hooks-reference.html#usecallback) : The React useCallback Hook returns a memoized callback function.
  + [useMemo](https://legacy.reactjs.org/docs/hooks-reference.html#usememo)

The useCallback and useMemo Hooks are similar. The main difference is that useMemo returns a memoized value and useCallback returns a memoized function

* + [useRef](https://legacy.reactjs.org/docs/hooks-reference.html#useref) : The useRef Hook allows you to persist values between renders.

It can be used to store a mutable value that does not cause a re-render when updated.

It can be used to access a DOM element directly.

useRef() only returns one item. It returns an Object called current.

When we initialize useRef we set the initial value: useRef(0)

1. React.createContext
2. Context.provider
3. Context.Consumer
4. Class.contextType

## **Pre-requisites for React Hooks**

1. Node version 6 or above
2. NPM version 5.2 or above
3. Create-react-app tool for running the React App

## **What is Sass**

Sass is a CSS pre-processor.

Sass files are executed on the server and sends CSS to the browser.

You can learn more about Sass in our [Sass Tutorial](https://www.w3schools.com/sass/default.php).

### React.PureComponent

React.PureComponent is similar to [React.Component](https://legacy.reactjs.org/docs/react-api.html" \l "reactcomponent). The difference between them is that [React.Component](https://legacy.reactjs.org/docs/react-api.html" \l "reactcomponent) doesn’t implement [shouldComponentUpdate()](https://legacy.reactjs.org/docs/react-component.html" \l "shouldcomponentupdate), but React.PureComponent implements it with a shallow prop and state comparison.