LEARN-REACT

A Guide For The Beginner

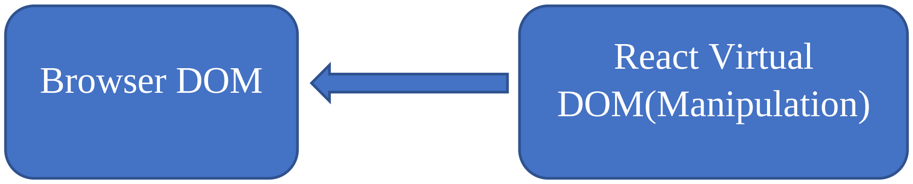
SHEERAZ MEHBOOB

Table of Contents

React JS

Introduction:

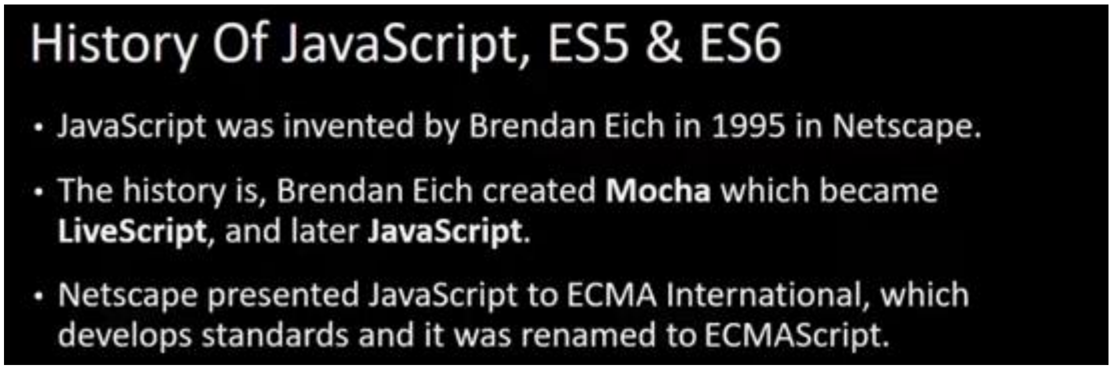
- Created by a Fb Software Engineer Jordan Walke.
- Library for building user interface.
- Allows to create reusable components.
- Free and open source library.
- Created by Facebook (2011 for fb Newsfeed feature).
- Released publically in July 2013(V0.3.0)
- Provides advance concepts like state management and routing etc.
- Follows Declarative Approach
- Means Abstraction
- Only Necessary Work is shown
- Tell browser what to do and react itself abstract unnecessary things.
- Creates Virtual DOM in its memory.



Create React App:

- Select Location
- Create-react-app <app name>
- Provides tools like babel, webpack, ESLint.

Babel:



History Of JavaScript, ES5 & ES6

- JavaScript was invented by Brendan Eich in 1995 in Netscape.
- The history is, Brendan Eich created **Mocha** which became **LiveScript**, and later **JavaScript**.
- Netscape presented JavaScript to ECMA International, which develops standards and it was renamed to ECMAScript.

- Babel is JS Compiler.
- Used to convert ES 2015+ code in backward compatible version of JS in all browsers.
- Not necessary to use only in React.
- Babel Can convert JSX Syntax.

Install Node.js & NPM:

- Download and install Node.js
- Check Version:
    - Node –v
    - Npm –v

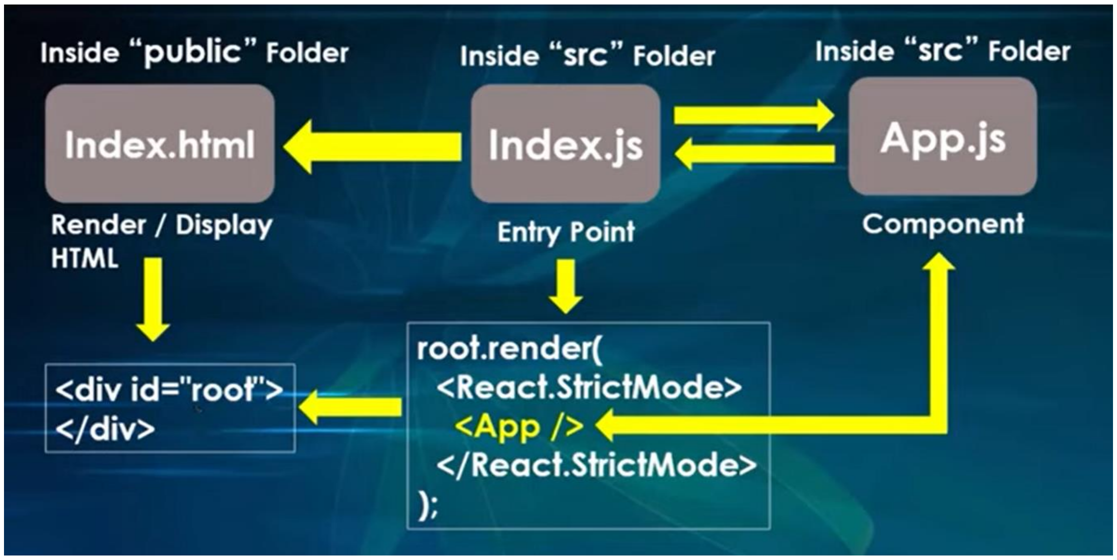
Install React:

- Npm install –g create-react-app
- Check Version:
    - Create-react-app –version

Create React App:

- Select Location
- Create-react-app <app name>
- Npm start

Workflow of React App:



React DOM:

- A function render is used to render HTML to web page.
- The purpose of the function is to display the specified HTML code inside the specified HTML element.
- Renders in index.html (in a div with id root).

JSX (JavaScript XML):

- In JSX we can directly return HTML.
- Follows XML rules.
- Use Camel Case for css and HTML attributes.
- To use images, we have to import them.
- An extension of Javascript.
- JSX code converted into babel JS.
- JSX convert HTML into react elements.
- Not compulsory to use in react but easy to use.
- Based on ES6 and converted in JS on runtime.
- We can also use JavaScript expression inside HTML code.
- We can say JSX is HTML code but we use JS expression, variables, objects etc. in {} curly braces.
- Can't use statements inside JSX but can use ternary operators.
- Example:

```
Function App ()

{

<div>

<h1>Hello</h1>

<p>My Name is Sherry</p>

</div>

}
```

One Top Level Element (Fragmentation):

- JSX only return one element. If we want to return multiple elements we can use Recat.Fragment or “<> </>”.

Why we import react module?

- JSX is combination of Html and JS code so to run this we need to import react module.
- JSX wouldn’t work without react module.

Components:

- Component name always start with capital letter.
- Independent and reusable code.
- Same as JS functions but work in isolation and return HTML.

Types:

1. **Class Component**
2. **Function Component**

Functional Components:

- Like functions that return HTML.
- Remember: Always use parentheses with return keyword because if you bring it to next line component will not returned.

Props (Properties) with Functional Components:

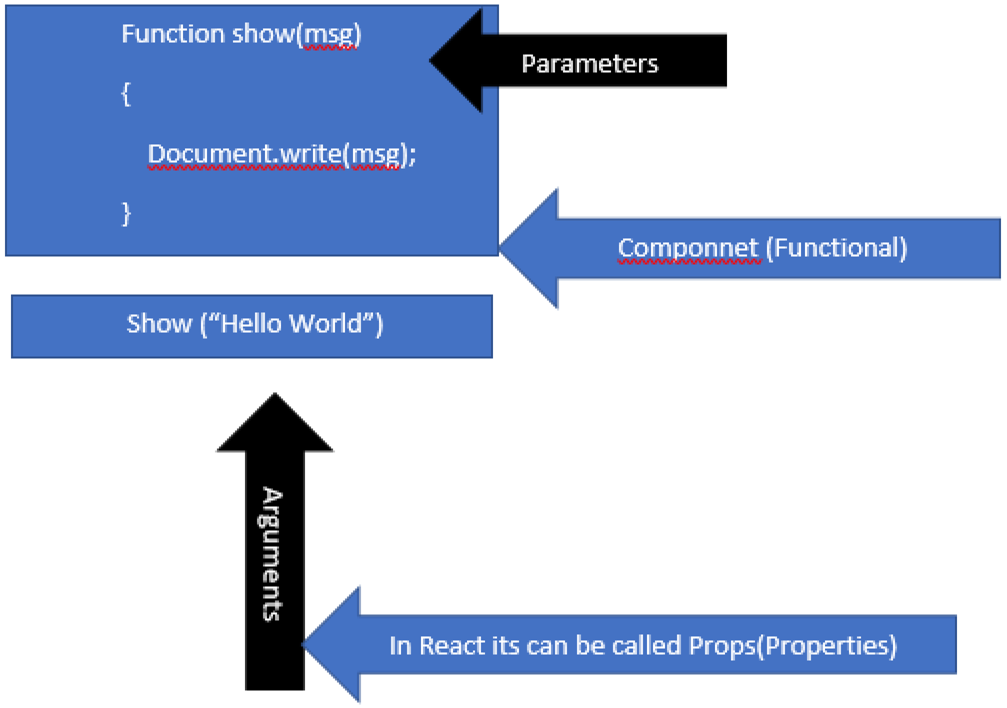
- Props are arguments passed in React Components.
- Props are like arguments in JS and attribute in HTML.
- Props work same as object.
- Can pass variable also but use curly braces as will be using JS inside HTML/JSX tags.
- React props are immutable or read only.
- Props can be used as children props.
- Example:

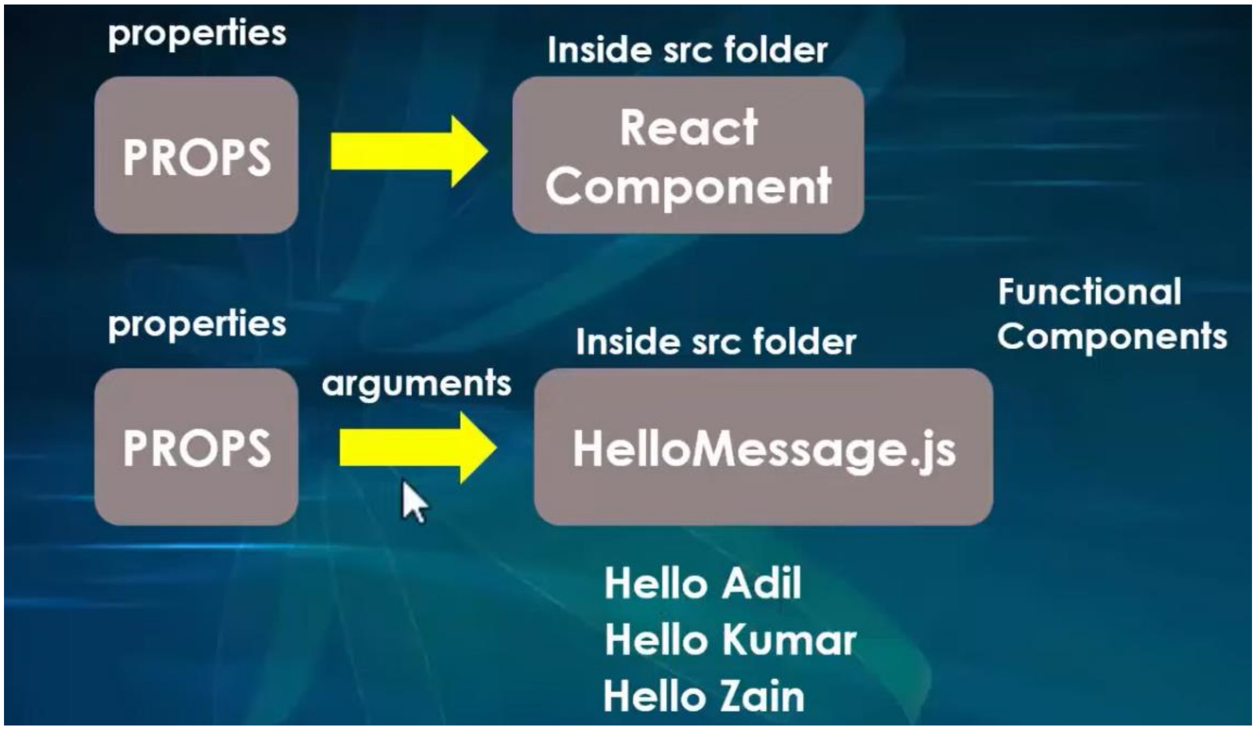
  When calling place anything as a child.

  Can be called as {props.children}

  This will render all children props elements on web page.

- Consider a simple function in JS
- Use to work dynamically, like print message dynamically.


Function show(msg)
{
Document.write(msg);
}
Parameters
Componnet (Functional)
Show ("Hello World")
Arguments
In React its can be called Props(Properties)


properties
PROPS
Inside src folder
React Component
properties
PROPS
arguments
Inside src folder
HelloMessage.js
Functional Components
Hello Adil
Hello Kumar
Hello Zain

Example of Props:

❖ **Component.js:**

```
Import React from ‘react’;

Function Component(props)

{

        Return (

                <h1>{props.name} </h1>

        );

}
```

❖ **App.js:**

- Import React from ‘react’;
- Import Comp from ‘. /Component’;
- Pass props when calling a component and access it inside that component.

```
Function App ()
{
Return  <” Component” name=” sherry” age= {19} />
}
```

Pass Data:

- Use to pass data from one component to other as parameters.

PropTypes -Type Checking with Functional Components:

- To install and use dependency for props
- Npm install prop-types
- Add Import: “import PropTypes from ‘prop-types’”
- To apply prop type checking on props we have to use built in PropTypes property.
- Before exporting a component where you have used prop types add

```
App.propTypes =
{
    Identifier: PropTypes. <datatype>,
    Identifier: PropTypes. <datatype>
}
```

- If we supply wrong data type data application will just give warning JS Console but will work.

- To validate we can use isRequired property so if we forgot to provide value it will give a warning.

```
App.propTypes =
{
        Identifier: PropTypes. <datatype>. isRequired,
        Identifier: PropTypes. <datatype>. isRequired
}
```

- To use Boolean values, you must have to convert it into string.
- We can also set default values for props.
- App.defaultProps =

```
{
        Identifier: ‘Hello’,
        Identifier: ‘18’
}
```

De-Structuring Props in Functional Components:

```
Function Component(props)
        {
                Return (
                        <h1>{props.name} </h1>
                        <h1> {props. Age} </h1>
                );
        }
```

Instead of above we can use:

```
Function Component (name, age)
        {
                Return (
                        <h1>{name}</h1>
                        <h1>{age}</h1>
                );
}
```

OR

Function Component(props)

```
        {
                Let {name, age} = props;
                Return (
                        <h1>{name}</h1>
                        <h1>{age}</h1>
                );
}
```

- Keep in mind variable names must be same that has been passed as parameters.

Advantages:

- Improve readability.
- Provides same data properties.
- Cut amount of code.
- Saves time from iterate over an array of object multiple times.

Class Components:

- These are simply classes containing multiple functions.
- All class components have a parent class Component Class which is in React Module (React.Component).
- Class component must implement render member function which returns a component to be rendered.
- Import and export of components is same.
- We can use both class and functional component at the same time also but default export can only be one.

Format:

```
class App extends React.Component
{
        Render ()
        {
                Return (
                        <h1>Hello World</h1>)
        }
}
```

Props with Class Components

- Same as in functional component.
- Access using this keyword like “this.props.name;”

PropTypes -Type Checking with Class Components:

- Same as Functional PropTypes

Constructors in React:

Member of class

Use to initialize the object

Automatically called when object is created and in react object is created during calling of object.

Constructor in React JS is a method used to initialize the states of object in a class.

We use super keyword inside child class constructor which will call parent constructor.

If we don’t initialize state or don’t bind methods, then no need to implement constructor for React Component.

We can’t use this keyword if we don’t call super method in constructor.

State with Class Component:

- Built-in react object used to contain data about component.
- State created and managed inside component.
- State is stored in the form of variables.
- States are mutable.
- When we update value of state component will be re-rendered.
- Syntax:
    - With Constructor

```
Constructor ()

{

        Super ();

        This.state= {

        Name=” sheeraz”

        }
```

}

- Without Constructor

  state= {

  Name=" sheeraz"

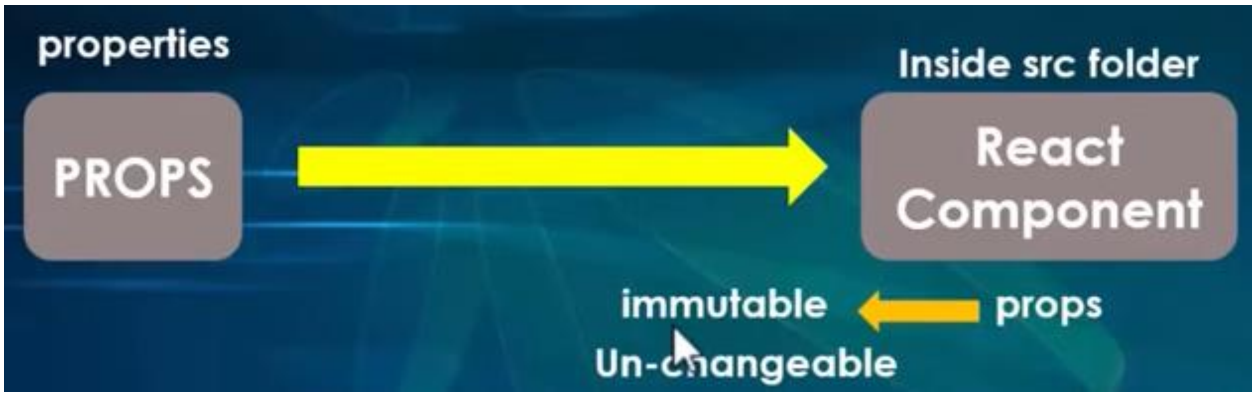
  }

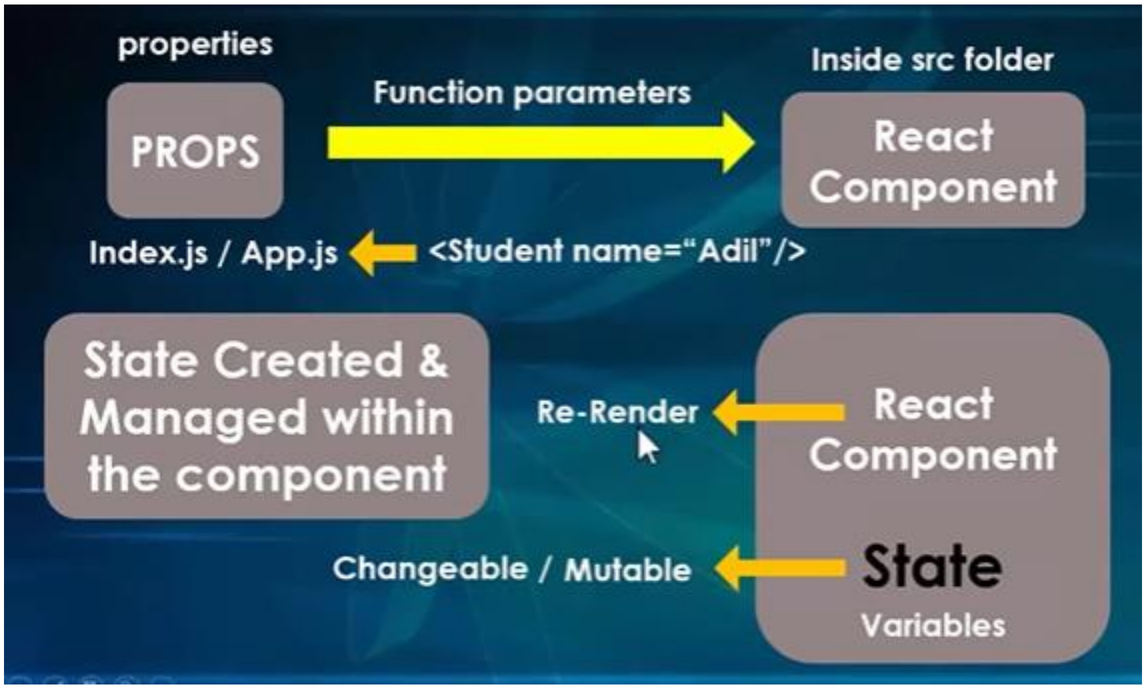
- To change the value of state we have to use a function named setState.

  setState ({

  name=" Malik"})

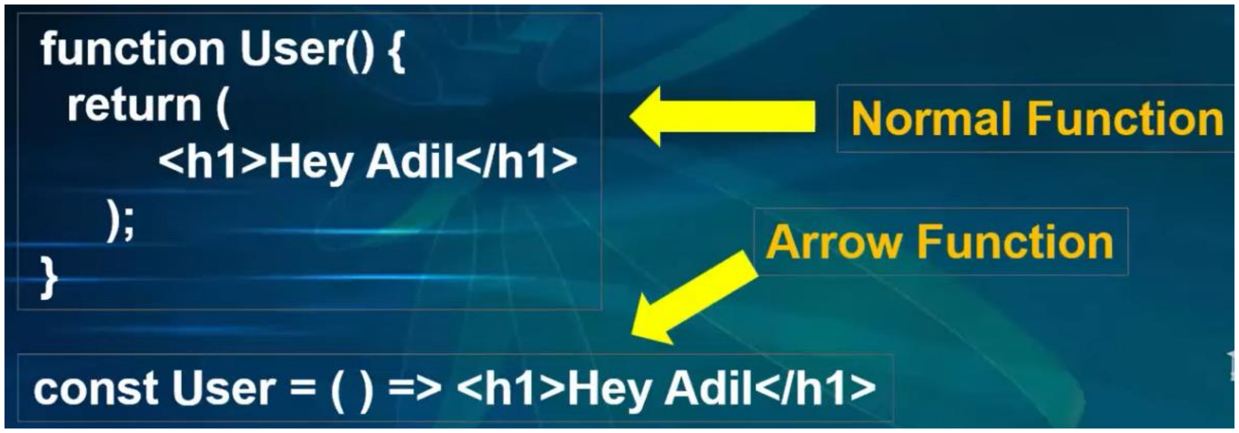
- State updated in response to event handlers, server responses, props changes.
- Only created in class components
- State object is initialized in constructor.
- We can set props data to state.





Arrow Functions:

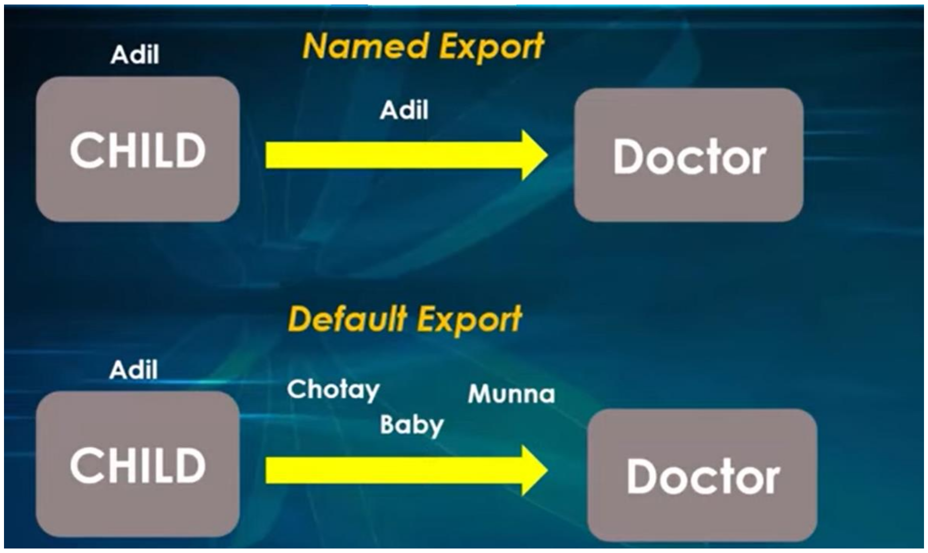
- Allows to write shorter function syntax.
- For one statement you can remove brackets and return keyword.
- Introduced in ES6.



```
function User() {
  return (
        <h1>Hey Adil</h1>
     );
}
```

```
const User = ( ) => <h1>Hey Adil</h1>
```

Default vs Named Export:



- Only one default export allowed per module while named can export many.
- In default we can use any alias while importing but in named names also must be same.
- Named Syntax:

  Export {App, App2, …};

  Import {App, App2, …} from '. /App';

- Default Syntax:

  Export default app;

  Import App from '. /App'

- We can also use both exports together but syntax will be:

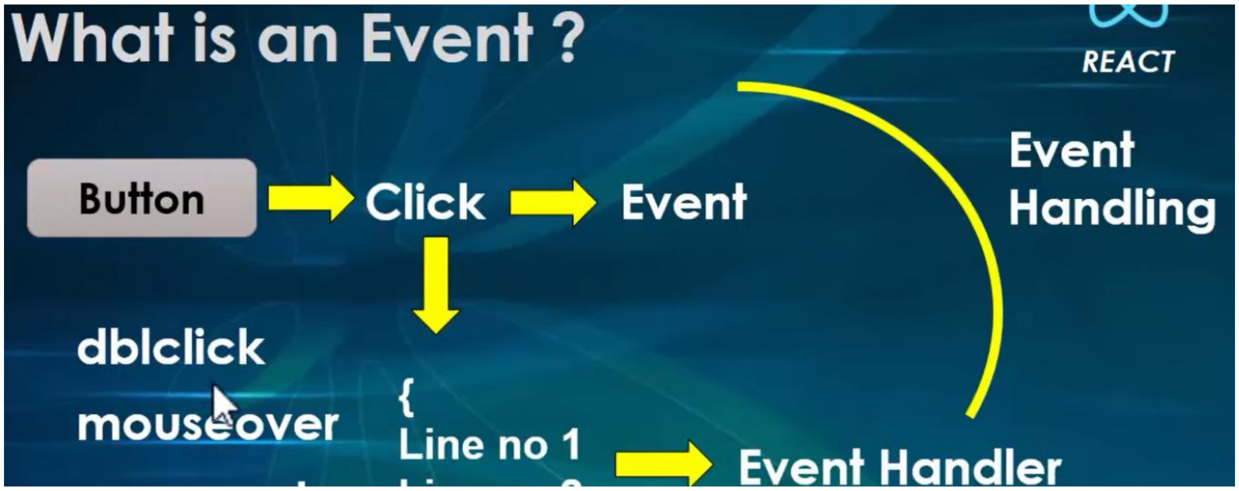
  Export default App

  Export {App2}

  Import App, {App2} from '. /App'

Events:

Events are actions that happen in our system and system tell us so our code can react to them.



Event handlers are written in curly braces. i.e. onClick={Function}

Passing Argument: To use an argument to an event handler use an arrow function.

Modern JS allows us to define a function inside another function.

```
const HelloFunction = (name) => {
        alert("Hello "+name)
    }

<input type="button" value="Argumented Click" onClick= {() => HelloFunction("Sherry")} />
```

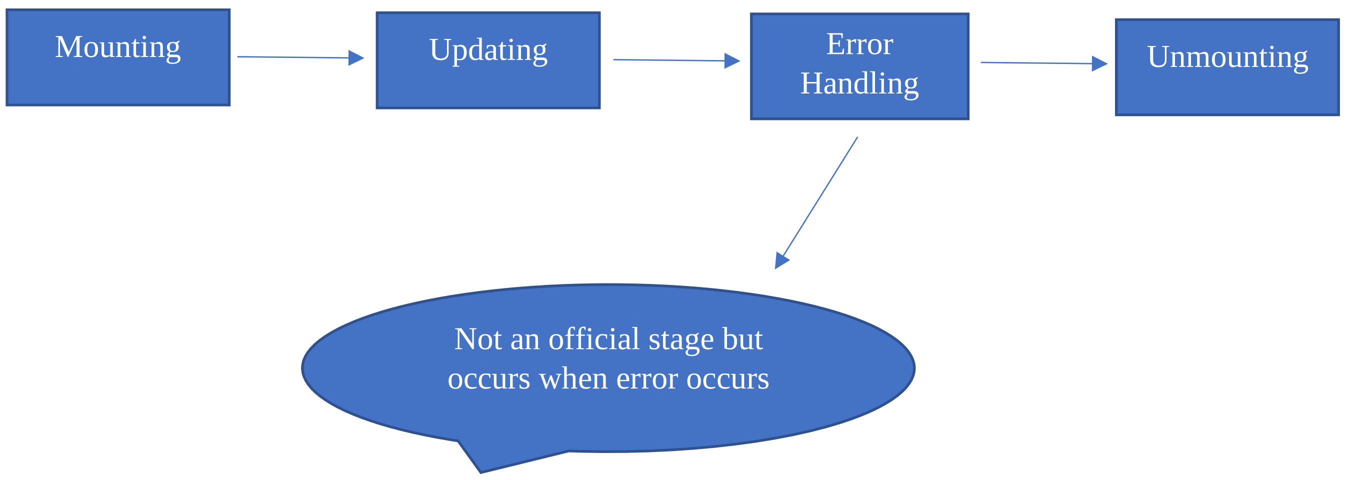
Remember: When calling function, no need to use () parentheses else function will be called quickly instead of waiting for events like click etc.

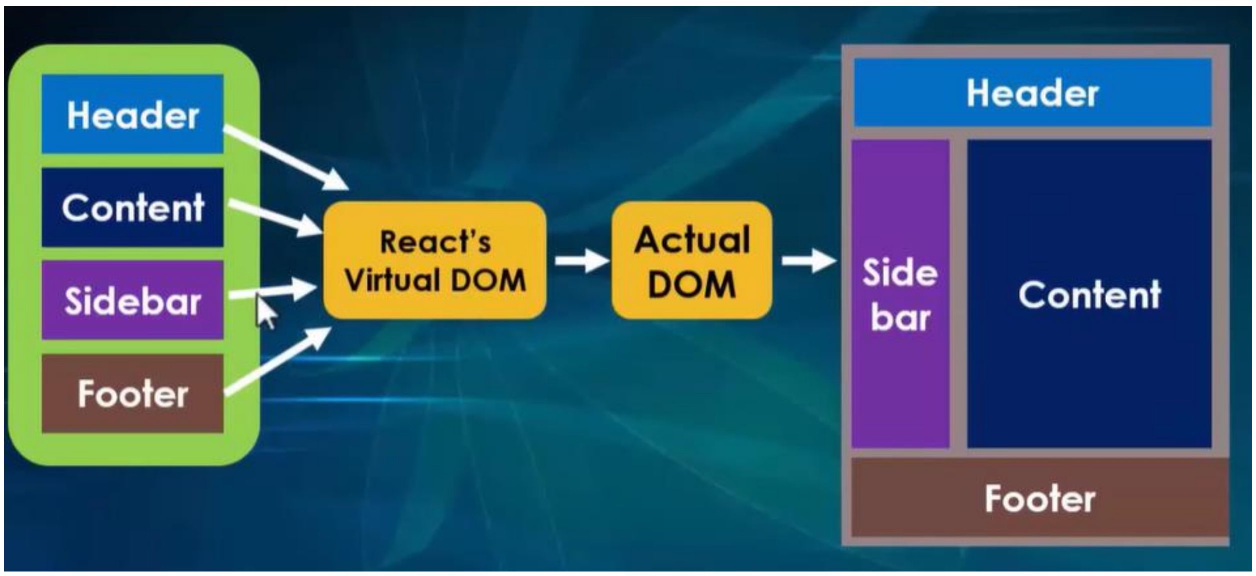
→ In React class component we need to bind event handler with function so we can use this keyword.

React Life Cycle of Components:

Each Component has these three phases:

- Mounting (Arrange, Setup, Prepare, Initialize)
    - Putting elements into the dom that we want to display in our react app.
- Updating
    - A component is updated whenever there is change in State or props.
- Unmounting (Un-initialize, Destroy)
    - Components/Elements are removed from the DOM.





→ These life cycles are only applied to class components but if we want to use it in functional component by using Hooks.

Conditional Rendering:

Render and hide whatever you want according to the conditions.

Ways to do this:

- If-else Statements:
  - Same as in JS
  - You can return component directly from condition as a statement or can store it in element variable and then call it in React return function as a JS statement inside {}
  - **1st Method**

```
function ConditioalRendring () {
const name="Sherry";
if(name=="Sherry”) {
    return <C1forConditionalRendering/>
}
else {
    return <C2forConditionalRendering/>
}
}
```

  - **2nd Method**

```
function ConditioalRendring () {
const name="Sherry";
let data;
if(name=="Sherry”) {
    data= <C1forConditionalRendering/>
}
else {
    data= <C2forConditionalRendering/>
}
return (
<>
    {data}
</>
);
}
```

- Logical && Operator:
- Ternary Operator

.

List/Array Rendering:

In a website we can use list to display list of names, products, courses etc.

In react we can use loops to render a list on web page but the JS map () is preferred.

Simple Array/List

```
function ListRendering () {
    const names=["Dawood","Sherry","Ali","Adil"];
  return (
    <div>
      {names.map (name => <h1> {"Hello, "+name} </h1>)}
    </div>
  )
}
```

Array of Objects:

List and keys in React:

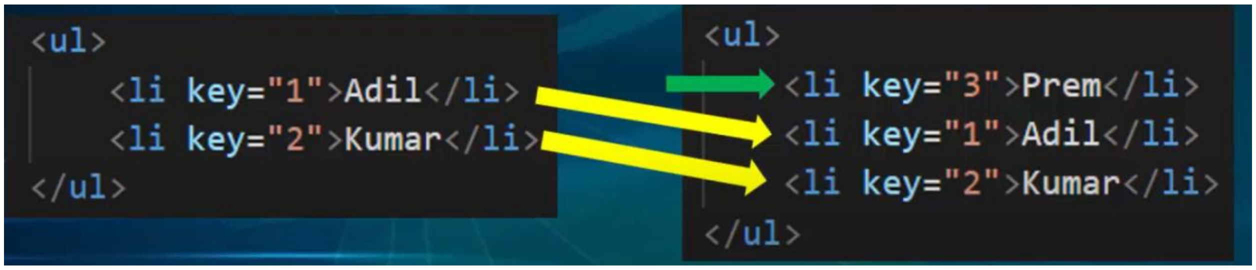
Why?

Each child in a list should have a unique key prop.

Developer does not need this key, React Uses it to keep track of elements and for updating it by comparing new element with previous using keys.

Using this way only newly added or removed item will be re-rendered.

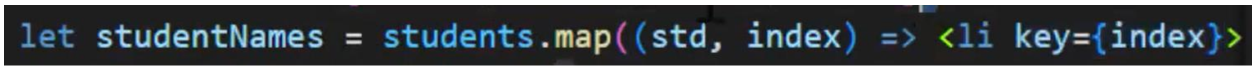
Keys should be unique.



```
<ul>
    <li key="1">Adil</li>
    <li key="2">Kumar</li>
</ul>
```

```
<ul>
    <li key="3">Prem</li>
    <li key="1">Adil</li>
    <li key="2">Kumar</li>
</ul>
```

Syntax:



```
let studentNames = students.map((std, index) => <li key={index}>
```

Hooks:

- Used in function based components.
- Allows to use state in functional components.
- Hooks are basically functions and allow us to hook react features like state and life cycle in functional components.

Rules to use Hooks:

- Can only be called in functional component.
- Hooks can be called at top level of component.
- Hooks should not call in loops, conditions and nested functions.
- Cannot be called from a local function of JS.
- Hooks relies on order in which they are called.

When to Use Hooks?

Earlier if we want to work with some state in functional component we have to convert it into a class component and then add state but now we can do it inside functional component using hooks.

Available Hooks:

- useState
- useEffect
- useContext
- useRef
- useReducer
- useCallback
- useMemo
- CustomHooks

useState:

useState function is used to set and retrieve state.

useState returns an array containing any primitive type of data.

When updating complete state is updated but if we want specific updating we can use spread operator.

Named import useState form react.

useState accepts an initial value and return two values:

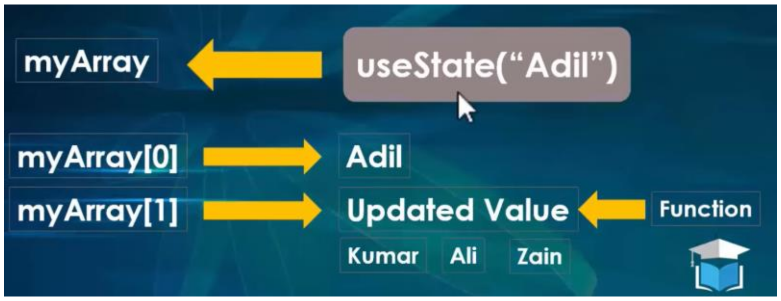
1- Current state
2- A function that update state

Spread Operator: Quickly copy all content of existing array or object into another array/object and return a single array/object.

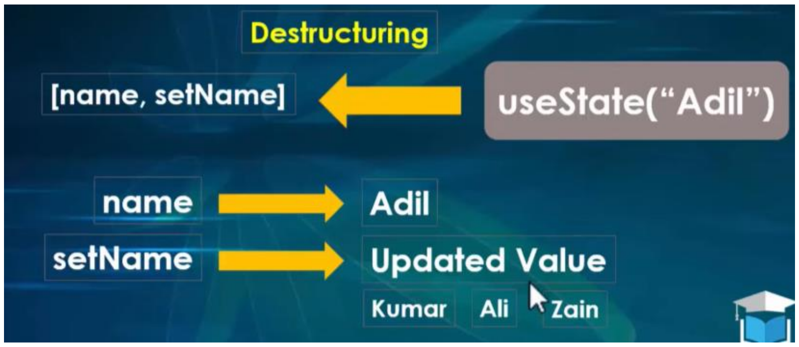
1st Method:

Arrays will have 2 indexes

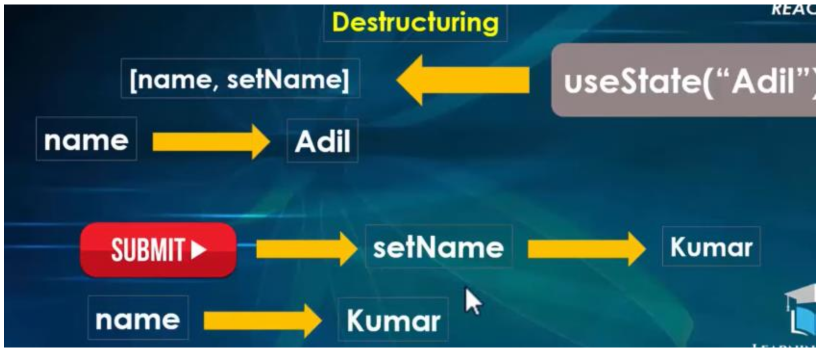
At array[0] we will have original data and at array[1] we will have a function use to update data.



2nd Method:



3rd Method:



useEffect:

What Is Effect?

Effect refers to a functional programming term “side effect”. Functional programming is a way of writing computer programs where we use functions to do things, and we try to make sure that our functions don't change anything else in the program or in the world around us. This is called a "side effect". We want to avoid side effects as much as possible because they can make our programs harder to understand and work with. But sometimes we still need to use side effects, like when we want to read from or write to a file.

Examples of side effects:

- Making a request to an API for data from a backend
- To interact with browser APIs (use document or window directly as in React we actually use Virtual DOM)
- Using unpredictable timing functions like setTimeout or setInterval.

Why we need useEffect?

useEffect Hook allow us to perform side effects in functional component. Sometimes we need to run some additional code after component has been rendered. Render () method shouldn’t cause side effects as it called earlier when currently not all elements been rendered.

Comparison:

We can compare useEffect hook with class component’s life cycle:

- Mounting
  - Mounting is like a constructor which is called when initializing an object so useEffect can also act as Mounting State.
- Updating
  - It is a method called whenever a state is updated in the same way useEffect can be called on updation of a state.
- Unmounting
  - It is more like a constructor which runs to un-initialize an object in same way useEffect can be used.

How to use?

- Import useEffect from react
  - import {useEffect} from ‘react’
- Place useEffect function inside your functional component as it will be easy to access props and states within the component.
- By default, useEffect runs after rendering and after every render update, however you can change it with the help of conditional useEffect concept.
- useEffect accepts two arguments but 2nd argument is optional:

- useEffect (<function>, <dependency>)
    - The function passed as first argument will run every time according the condition.
    - Second argument is dependent on first. It is the array of values which depends on the effects. This array will specify the conditions for function to run.

- Remember: You can call useEffect function as many times as you need.

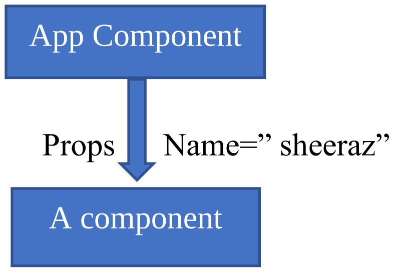
useContext:

Introduction:

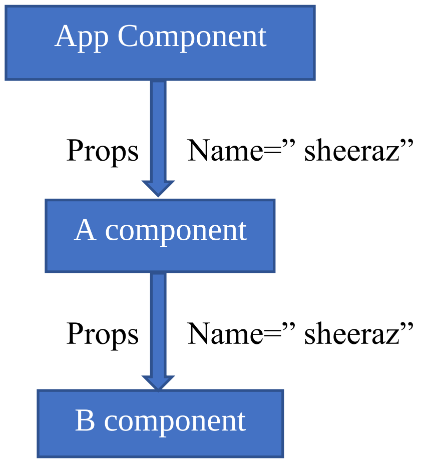
- React context is a way to manage state globally.
- Can be used with useState hook to share state between deeply nested components more easily.

Why we use useContext Hook?

For passing data from App component to any component in 1 level we use props. E.g.



But consider the scenario that you want to pass data to B component only but if you use props you must first pass it to A component and then A will pass data as props to B as B is a nested component. This is called prop drilling that has disadvantage as we don't need Name in A component but we have to pass.



So, if we want to send data directly to a component without data drilling we will use useContext Hook.

- We can create as many contexts as needed.
- We can also pass state inside a context.
- We will only use prop drilling when we need data in every component.

How to use it?

- To find the method to use it goto src folder find ‘App.js’ then components and find for
    - ‘UseContextComponentA’
    - ‘UseContextComponentB’
    - ‘UseContextComponentC’
- You can see the proper implementation to use useContext Hook.