React Fundamentals

Installation

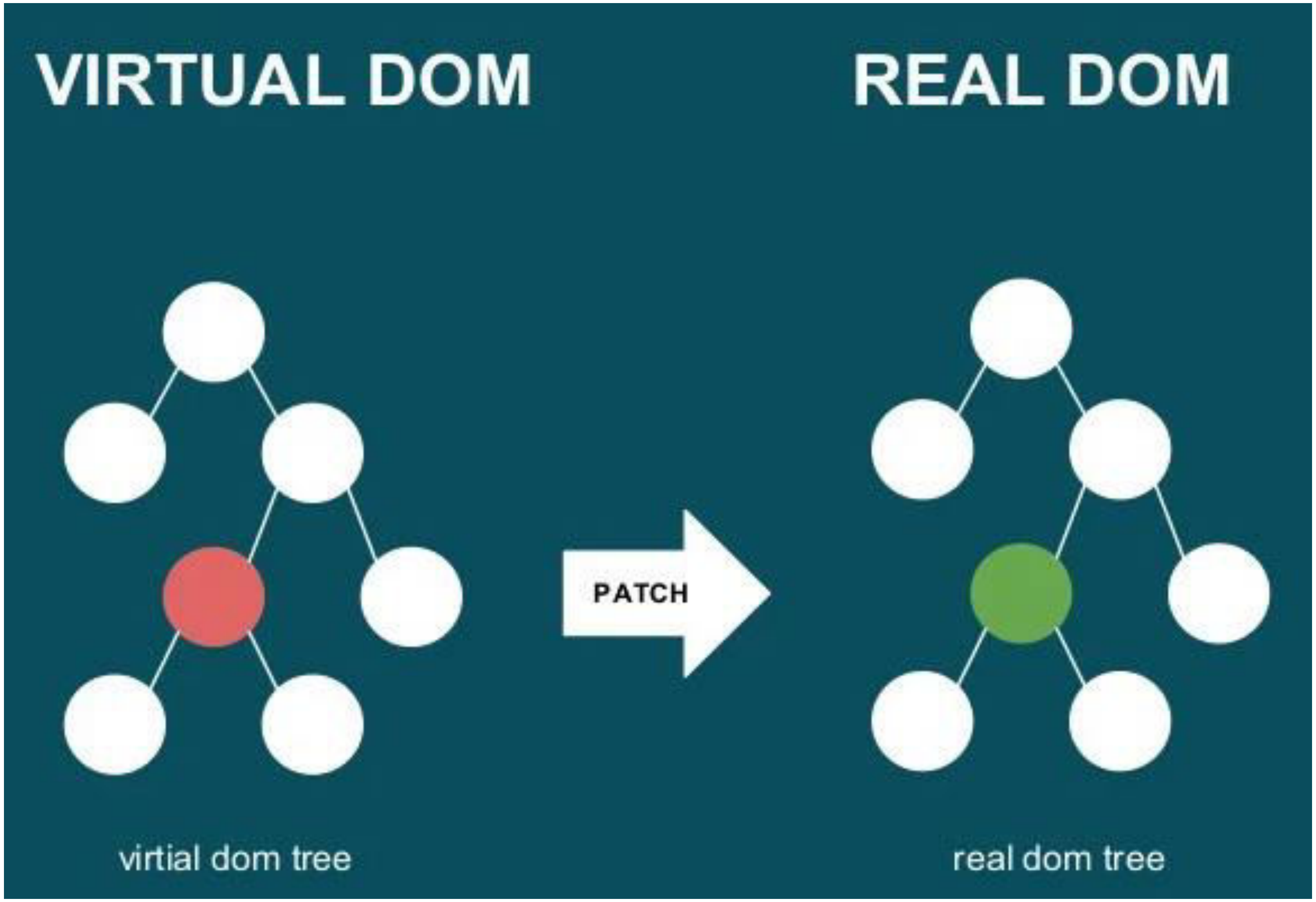
- NodeJS is the platform needed for the ReactJS development.
- Visual Studio Code Editor

React JS

- ReactJS is a **declarative**, **efficient**, and flexible **JavaScript library** for building reusable UI components.
- It is an open-source, component-based front end library which is responsible only for the view layer of the application.
- It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.

Why use ReactJS?

- The main objective of ReactJS is to develop User Interfaces (UI) that improves the speed of the apps.
- It uses virtual DOM (JavaScript object), which improves the performance of the app.
- The JavaScript virtual DOM is faster than the regular DOM.
- We can use ReactJS on the client and server-side as well as with other frameworks.
- It uses component and data patterns that improve readability and helps to maintain larger apps.


VIRTUAL DOM
REAL DOM
PATCH
virtial dom tree
real dom tree

How does React Work?

- React creates a VIRTUAL DOM in memory.
- 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.
- React only changes what needs to be changed!
- React finds out what changes have been made, and changes **only** what needs to be changed.

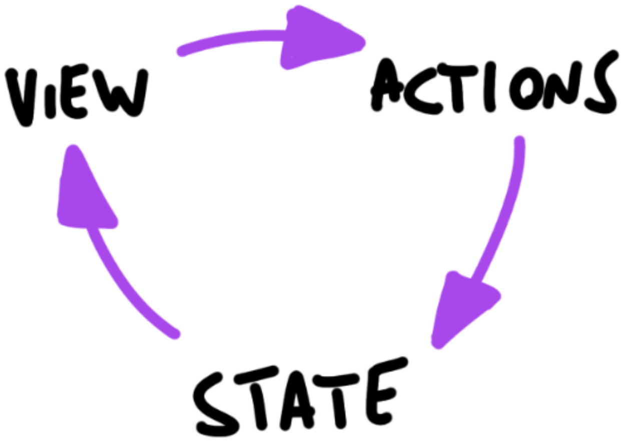
Note

- ReactJS offers graceful solutions to some of front-end programming’s most persistent issues, allowing you to build dynamic and interactive web apps with ease.
- It’s fast, scalable, flexible, powerful, and has a robust developer community that’s rapidly growing.

React Features

- **JSX** − JSX (Javascript XML) is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.
- **Components** − React is all about components. You need to think of everything as a component. This will help you maintain the code when working on larger scale projects.
- **Unidirectional data flow and Flux** − React implements one-way data flow which makes it easy to reason about your app. Flux is a pattern that helps keeping your data

Unidirectional data flo



- In React this means that:
- state is passed to the view and to child components
- actions are triggered by the view
- actions can update the state
- the state change is passed to the view and to child components

Advantage of ReactJS

- Easy to Learn and Use
- Creating Dynamic Web Applications Becomes Easier
- Reusable Components
- Performance Enhancement
- The Support of Handy Tools
- Known to be SEO Friendly
- The Benefit of Having JavaScript Library
- Scope for Testing the Codes

Disadvantage of ReactJS

- The high pace of development
- Poor Documentation
- View Part
- JSX as a barrier

React Environment Setup

- There are two ways to set up an environment for successful ReactJS application.
- Using the npm command
- Using the create-react-app command

Using the npm command

- **Install NodeJS and NPM**
- NodeJS and NPM are the platforms need to develop any ReactJS application.
- You can install NodeJS and NPM package manager by the link given below.

- https://nodejs.org/en/download/
- To check if properly installed in cmd / terminal
- node –v
- npm –v

Using npm

- **Install React and React DOM**
- Create a **root** folder with the name **reactApp** on the desktop
- Now, need to create a **package.json** file. To create any module, it is required to generate a package.json file in the project folder.
- npm init -y
- After creating a package.json file, you need to install **react** and its DOM **packages** using the following npm command in the terminal window

- npm install react react-dom --save

- You can also use the above command separately which can be shown as below.
- npm install react --save
- npm install react-dom --save
- **Install Webpack**
- *Webpack is used for module packaging, development, and production pipeline automation.*
- *will use **webpack-dev-server** during development,*
- ***webpack** to create production builds, and*
- ***webpack CLI** provides a set of commands.*

Webpack

- npm install webpack webpack-dev-server webpack-cli --save
- Or
- npm install webpack --save
- npm install webpack-dev-server --save
- npm install webpack-cli --save

- **Install Babel**
- *Babel is a JavaScript compiler and transpiler used to convert one source code to others. It compiles React JSX and ES6 to ES5 JavaScript which can be run on all browsers.*
- npm install babel-core --save-dev
- npm install babel-loader --save-dev
- npm install babel-preset-env --save-dev
- npm install babel-preset-react --save-dev
- npm install babel-webpack-plugin --save-dev

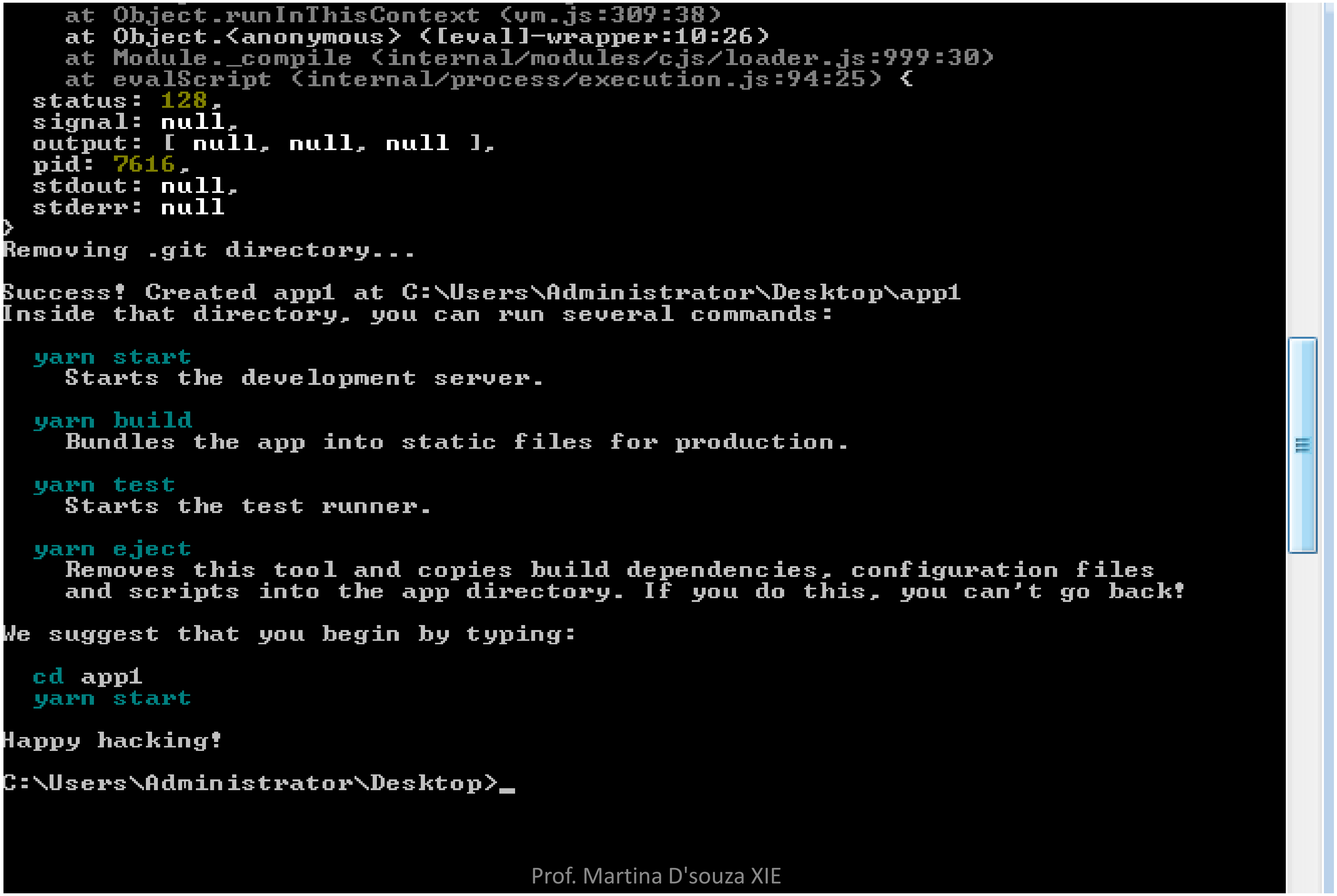
- To complete the installation process, need to add the following files in your project folder.
- These files are **index.html**, **App.js, main.js, webpack.config.js** and, **.babelrc.**

Using the create-react-app command

- *The 'create-react-app' is a tool maintained by Facebook itself.*
- *This is suitable for beginners without manually having to deal with transpiling tools like webpack and babel.*
- The create-react-app is an officially supported way to create React applications.
- If you have NPM and Node.js installed, you can create a React application by first installing the create-react-app.

- Install create-react-app by running this command in your terminal / cmd:
- npm install -g create-react-app
- Create a new React project
- create-react-app folder name
- **OR**
- We can combine the above two steps in a single command using npx.
- The npx is a package runner tool that comes with npm 5.2 and above version.
- C:\Users\Your Name>npx create-react-app folderName

cmd



```
    at Object.runInThisContext (vm.js:309:38)
    at Object.<anonymous> ([eval]-wrapper:10:26)
    at Module._compile (internal/modules/cjs/loader.js:999:30)
    at evalScript (internal/process/execution.js:94:25) {
  status: 128,
  signal: null,
  output: [ null, null, null ],
  pid: 7616,
  stdout: null,
  stderr: null
}
Removing .git directory...

Success! Created app1 at C:\Users\Administrator\Desktop\app1
Inside that directory, you can run several commands:

  yarn start
    Starts the development server.

  yarn build
    Bundles the app into static files for production.

  yarn test
    Starts the test runner.

  yarn eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd app1
  yarn start

Happy hacking!

C:\Users\Administrator\Desktop>_
```

- The above command will install the react and create a new project with the name foldername This app contains the following sub-folders and files by default which can be shown in the below image.

Run the React Application

- you are ready to run your first *real* React application.
- Run this command to move to the FolderName directory:
- C:\Users\Your Name> cd FolderName
- Run this command to execute the React application FolderName:
- C:\Users\Your Name\FolderName> npm start
- A new browser window will pop up with

Files and Folders in the root directory

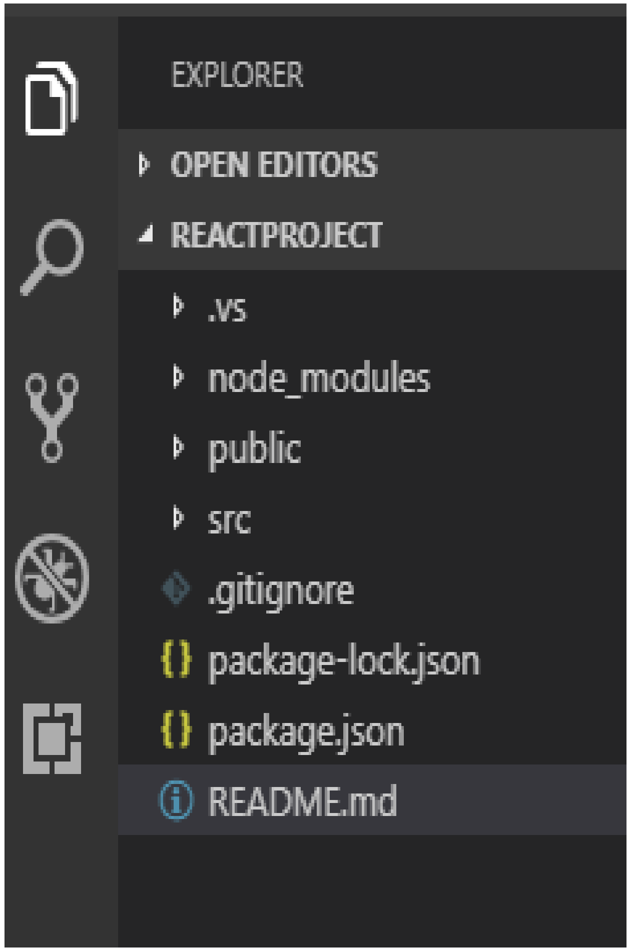
- **node_modules:** It contains the React library and any other third party libraries needed.
- **public:** It holds the public assets of the application. It contains the index.html where React will mount the application by default on the <div id="root"></div> element.
- **src:** It contains the App.css, App.js, App.test.js, index.css, index.js, and serviceWorker.js files.
- Here, the App.js file always responsible for displaying the output screen in React.

- **package-lock.json:** It is generated automatically for any operations where npm package modifies either the node_modules tree or package.json.
- **package.json:** It holds various metadata required for the project.
- It gives information to npm, which allows to identify the project as well as handle the projects dependencies.
- **README.md:** It provides the documentation to read about React topics.

create-react-app

- Create React App CLI tool removes all that complexities and makes React app simple.
- The **create-react-app** is an excellent tool for beginners, which allows to create and run React project very quickly.
- It does not take any configuration manually.
- This tool is wrapping all of the required dependencies like **Webpack**, **Babel** for React project itself and then you need to focus on writing React code only.
- This tool sets up the development environment provides an excellent

- open the project on Code editor.
- Eg. Visual Studio Code. project's default structure looks like as below image.


EXPLORER
OPEN EDITORS
REACTPROJECT
.vs
node_modules
public
src
.gitignore
package-lock.json
package.json
README.md

React Render HTML

- React's goal is in many ways to render HTML in a web page.
- React renders HTML to the web page by using a function called ReactDOM.render().
- The Render Function
- The ReactDOM.render() function takes two arguments, HTML code and an HTML element.
- The purpose of the function is to display the specified HTML code inside the specified HTML element.

```
<div id="root">
  <!-- This div's content will be managed by React. -->
</div>
```

We call this a “root” DOM node because everything inside it will be managed by React DOM.

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

JSX

- const ele = <h1>This is sample JSX</h1>;

- The above code snippet somewhat looks like HTML and it also uses a JavaScript-like variable but is neither HTML nor JavaScript, it is JSX.
- JSX is basically a syntax extension of regular JavaScript and is used to create React elements.
- These elements are then rendered to the

What is React JSX?

- JSX stands for JavaScript XML.
- JSX allows us to write HTML in React.
- JSX makes it easier to write and add HTML in React.
- JSX(JavaScript Extension), is a React extension which allows writing JavaScript code that looks like HTML.
- In other words, JSX is an HTML-like syntax used by React that extends ECMAScript so that **HTML-like** syntax can co-exist with JavaScript/React code.

JSX Advantage

- It is faster than normal JavaScript as it performs optimizations while translating to regular JavaScript.
- It makes it easier for us to create templates.
- Instead of separating the markup and logic in separated files, React uses *components* for this purpose.

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
const name = "Learner";
const element = <h1>Hello,{ name }.Welcome to
   XIE.< /h1>;
ReactDOM.render(element,
        document.getElementById("root")
);
```

Index.js

- Below is the example where conditional expressing is embedded in JSX:

```
import React from 'react';
import ReactDOM from 'react-dom';
let i = 1;
const element = <h1>{ (i == 1) ? 'Hello World!' :
   'False!' } < /h1>;
ReactDOM.render(
   element,
   document.getElementById("root")
);
```

Attributes in JSX

- JSX allows us to use attributes with the HTML elements just like with normal HTML.
- But instead of the normal naming convention of HTML, JSX uses camelcase convention for attributes.
- For example, *class* in HTML becomes *className* in JSX.
- The main reason behind this is that some attribute names in HTML like ‘class’ are reserved keywords in JavaScripts.
- So, in order to avoid this problem, JSX uses

- We can also use custom attributes in JSX.
- For custom attributes, the names of such attributes should be prefixed by **data-**.
- In the below example, we have used a custom attribute with name **data-sampleAttribute** for the <h2> tag.

```
import React from 'react';
import ReactDOM from 'react-dom';
  const element = <div><h1 className =
   "hello">Hello TEIT</h1>
            <h2 data-
   sampleAttribute="sample">Custom
   attribute</h2>< /div>;
   ReactDOM.render(element, document.getEle
   mentById("root")
);
```

- **Specifying attribute values**: JSX allows us to specify attribute values in two ways:
- **As for string literals:** We can specify the values of attributes as hard-coded strings using quotes:
- const ele = <h1 className = "firstAttribute">Hello!</h1>;
- **As expressions:** We can specify attributes as expressions using curly braces {}:
- const ele = <h1 className = {varName}>Hello!</h1>;

Wrapping elements or Children in JSX

- Consider a situation where you want to render multiple tags at a time.
- To do this we need to wrap all of this tag under a parent tag and then render this parent element to the HTML.
- All the subtags are called child tags or children of this parent element.

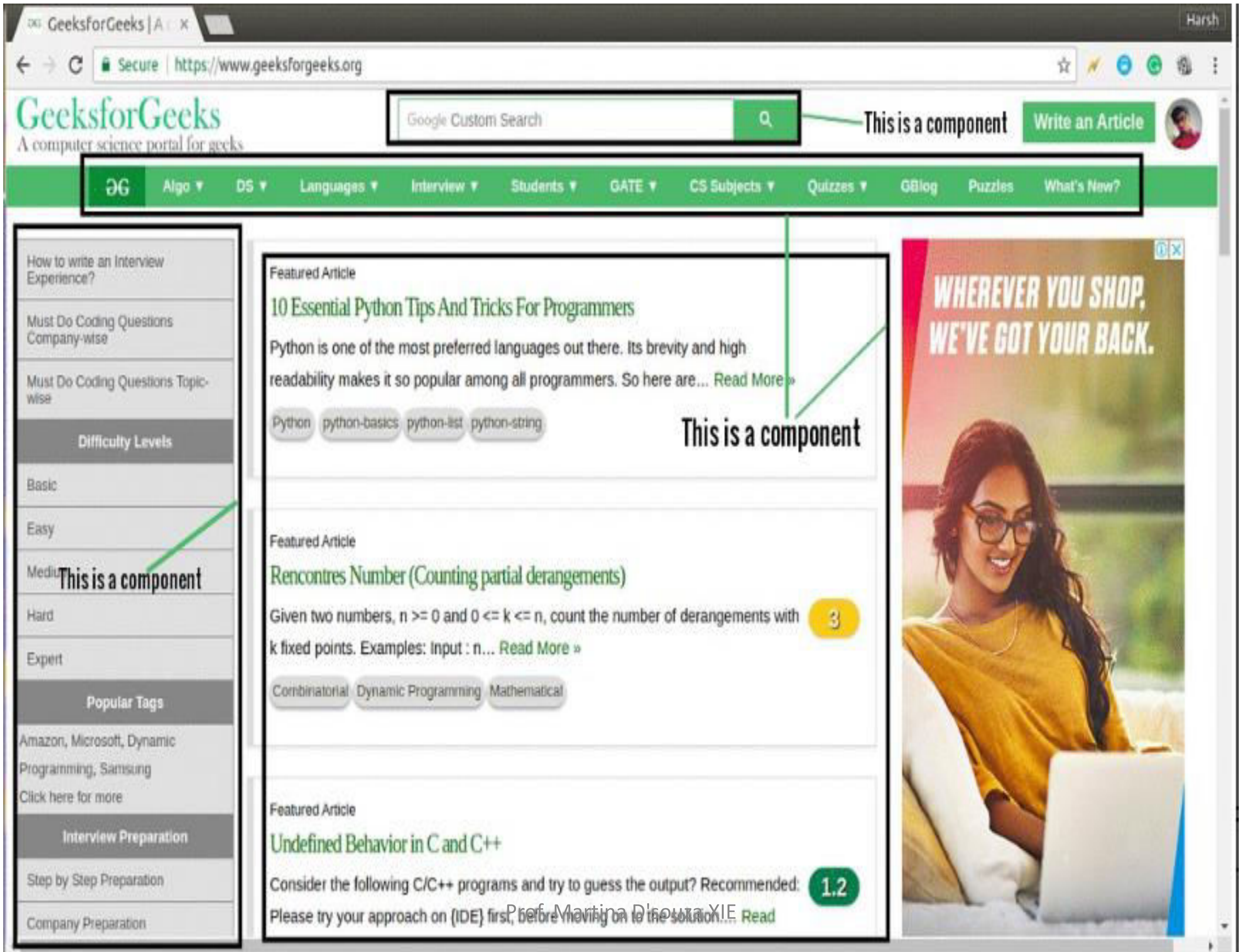
Index.js

```
    import React from 'react';
    import ReactDOM from 'react-dom';

const element = <div>
                    <h1>This is Heading 1 </h1>
                    <h2>This is Heading 2</h2 >
                    <h3>This is Heading 3 < /h3>
                    </div > ;
ReactDOM.render(element,
  document.getElementById("root"));
```

Components

- Components are bits of code that are reusable.
- They serve a similar purpose as JavaScript functions.
- A Component is one of the core building blocks of React.
- Every application develop in React will be made up of pieces called components.
- Components make the task of building UIs much easier.
- Eg; see a UI broken down into multiple


GeeksforGeeks | A
Harsh
Secure | https://www.geeksforgeeks.org
GeeksforGeeks
A computer science portal for geeks
Google Custom Search
This is a component
Write an Article
Algo ▾ DS ▾ Languages ▾ Interview ▾ Students ▾ GATE ▾ CS Subjects ▾ Quizzes ▾ GBlog Puzzles What's New?
How to write an Interview Experience?
Must Do Coding Questions Company-wise
Must Do Coding Questions Topic-wise
Difficulty Levels
Basic
Easy
Medium
Hard
Expert
Popular Tags
Amazon, Microsoft, Dynamic Programming, Samsung
Click here for more
Interview Preparation
Step by Step Preparation
Company Preparation
This is a component
This is a component
WHEREVER YOU SHOP, WE'VE GOT YOUR BACK.

Featured Article

10 Essential Python Tips And Tricks For Programmers

Python is one of the most preferred languages out there. Its brevity and high readability makes it so popular among all programmers. So here are... Read More »

Python python-basics python-list python-string

Featured Article

Rencontres Number (Counting partial derangements)

Given two numbers, n >= 0 and 0 <= k <= n, count the number of derangements with k fixed points. Examples: Input : n... Read More »

3

Combinatorial Dynamic Programming Mathematical

Featured Article

Undefined Behavior in C and C++

Consider the following C/C++ programs and try to guess the output? Recommended: Please try your approach on {IDE} first, before moving on to the solution.... Read

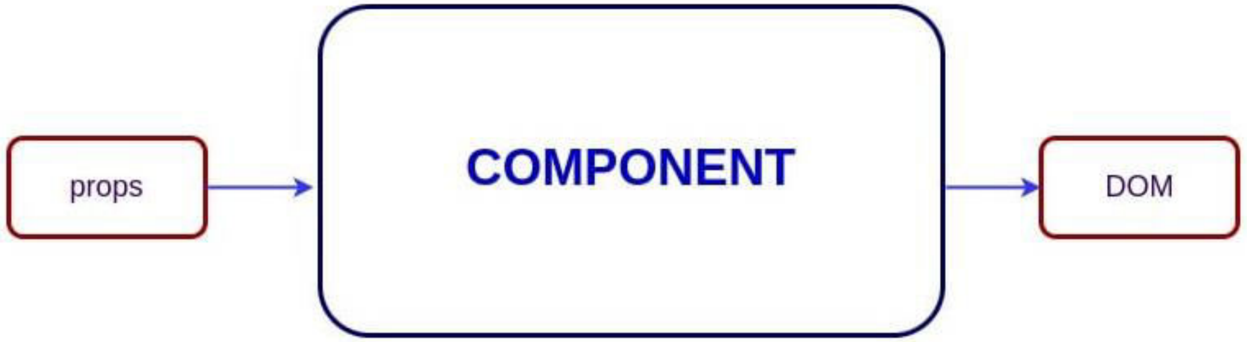
1.2

Props and State

- Components need data to work with.
- There are two different ways that can combine components and data:
- either as **props** or **state**.
- props and state determine what a component renders and how it behaves.

Props

- If components were plain JavaScript functions, then props would be the function input.
- Going by that analogy, a component accepts an input (what we call props), processes it, and then renders some JSX code.



Props

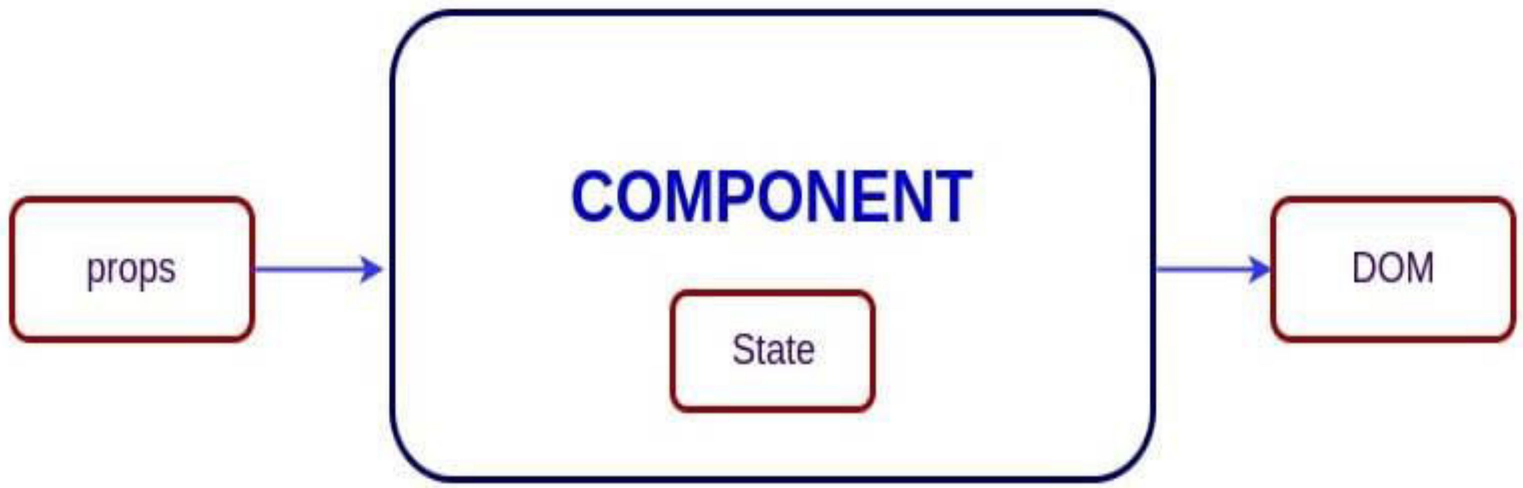
- props should be immutable and top-down.
- Means that a parent component can pass on whatever data it wants to its children as props, but the child component cannot modify its props.

ReactJS Props

- Props are nothing but properties to use inside a component.
- They are read-only variables that store the value of attributes of a tag.
- Props are similar to function arguments because you can pass them to the components in a similar way as arguments.
- You cannot modify props from inside the components.
- Instead, you can add attributes called props.

State

- State, is an object that is owned by the component where it is declared.
- Its scope is limited to the current component.
- A component can initialize its state and update it whenever necessary.
- The state of the parent component usually ends up being props of the child component.
- When the state is passed out of the current scope, refer to it as a prop.


props
COMPONENT
State
DOM

React Components

- Components are like functions that return HTML elements.
- React Components
- Components are independent and reusable bits of code.
- Components come in two types,
- Functional components,
- Class components .

Functional Components

- In React, function components are a way to write components that only contain a render method and don't have their own state.
- They are simply JavaScript functions that may or may not receive data as parameters.
- We can create a function that takes props(properties) as input and returns what should be rendered.
- A valid functional component can be shown in the below example.

```
function WelcomeMessage(props) {
  return <h1>Welcome to the ,
   {props.name}</h1>;
}
```

- This function is a valid React component because it accepts a single "props" (which stands for properties) object argument with data and returns a React element.
- We call such components "function components" because they are literally JavaScript functions.
- The functional component is also known as a

Class Components

- Class components are more complex than functional components.
- It requires to extend from React.Component and create a render function which returns a React element.
- Can pass data from one class to other class components.
- Can create a class by defining a class that extends Component and has a render function.
- Valid class component is shown in the below example

ES6 class to define a component:

```
class Welcome extends React.Component {
render() {
return <h1>Hello, {this.props.name}</h1>; }
 }
```

- The class component is also known as a stateful component because they can hold or manage local state.

- **Note: Always start component names with a capital letter.**
- React treats components starting with lowercase letters as DOM tags.
- For example, <div /> represents an HTML div tag, but
- <Welcome /> represents a component and requires Welcome to be in scope.

Rendering a Component

- Elements can also represent user-defined components:
- const element = <Welcome name="Sara" />;
- When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a single object.
- We call this object “props”.
- For example, this code renders “Hello, Sara” on the page:

Index.js

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
const element = <Welcome name="Sara" />;
ReactDOM.render(element,
   document.getElementById('root')
);
```

Note

- We call ReactDOM.render() with the <Welcome name="Sara" /> element.
- React calls the Welcome component with {name: 'Sara'} as the props.
- Our Welcome component returns a <h1>Hello, Sara</h1> element as the result.
- React DOM efficiently updates the DOM to match <h1>Hello, Sara</h1>.

- Components can refer to other components in their output.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Steve" />
      <Welcome name="Sonam" />
```

```
ReactDOM.render(
  <App />,

document.getElementById('root')
);
```

ReactJS State

- In ReactJS, the state contains information about the components and can change over time.
- It can either be a response to user action or a system event. Data collected in a state is a private object.
- Components within the state are called stateful components.
- They regulate the component's behavior and make it more interactive.
- You can access or modify the state only inside the component or directly by the

- You should always simplify your state as much as possible and reduce the number of stateful components.
- For example, if you have ten components requiring data from the state, you should create a single container component that will keep the state for all the components.

```
class Test extends React.Component {
   constructor() {
      this.state = {
         id: 1,
         name: "test"
      };
   }

   render() {
      return (
         <div>
           <p>{this.state.id}</p>
           <p>{this.state.name}</p>
         </div>
      );
   }
}
```

setState()

- State should not be modified directly, but it can be modified with a special method called setState( ).

- this.state.id = “2020”; // wrong

- this.setState({   // correct
-     id: "2020"
- });

State Change

- A change in the state happens based on user-input, triggering an event, and so on.
- Also, React components (with state) are rendered based on the data in the state.
- State holds the initial information.
- So when state changes, React gets informed and immediately re-renders the DOM – **not the whole DOM, but only the component with the updated state.**
- This is one of the reasons why React is fast.

Differences between props and state

- Components receive data from outside with props, whereas they can create and manage their own data with state
- Props are used to pass data, whereas state is for managing data
- Data from props is read-only, and cannot be modified by a component that is receiving it from outside
- State data can be modified by its own component, but is private (cannot be accessed from outside)
- Props can only be passed from parent

Note

- State could only be used in **class components**, not in functional components.
- That's why functional components were also known as stateless components.
- However, after the introduction of **React Hooks**, state can now be used both in class and functional components.

App.jsx

```
import React from 'react';
class App extends React.Component {
  constructor(props) {
    super(props);
        this.state = {
      header: "Header from state...",
      content: "Content from state..."
    }
  }
  render() {
    return (
      <div>
        <h1>{this.state.header}</h1>
        <h2>{this.state.content}</h2>
      </div>
    );   }}
export default App;
```

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App />,
   document.getElementById('app'));
```

props

- The main difference between state and props is that **props** are immutable.
- This is why the container component should define the state that can be updated and changed, while the child components should only pass data from the state using props.
- Using Props
- When we need immutable data in our component, we can just add props to **reactDOM.render()** function in **main.js** and use it inside our component.

App.jsx

```
import React from 'react';

class App extends React.Component {
   render() {
      return (
         <div>
            <h1>{this.props.headerProp}</h1>
            <h2>{this.props.contentProp}</h2>
         </div>
      );
   }
}
export default App;
```

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App headerProp = "Header from
    props..." contentProp = "Content
   from props..."/>, document.getElementById('app'));

export default App;
```

React Component Life-Cycle

- In ReactJS, every component creation process involves various lifecycle methods.
- These lifecycle methods are termed as component's lifecycle.
- These lifecycle methods are not very complicated and called at various points during a component's life.
- The lifecycle of the component is divided into **four phases**.
- Initial Phase
- Mounting Phase
- Updating Phase

1. Initial Phase

- It is the **birth** phase of the lifecycle of a ReactJS component.
- Component starts its journey on a way to the DOM.
- In this phase, a component contains the default Props and initial State.
- These default properties are done in the constructor of a component.
- The initial phase only occurs once and consists of the following methods.
- **getDefaultProps()**
  It is used to specify the default value of this.props.
- It is invoked before the creation of the component or any props from the parent is passed into it.
- **getInitialState()**
  It is used to specify the default value of this.state.
- It is invoked before the creation of the component.

2. Mounting Phase

- In this phase, the instance of a component is created and inserted into the DOM.
- It consists of the following methods.
  - **componentWillMount()**

  This is invoked immediately before a component gets rendered into the DOM. In the case, when you call **setState()** inside this method, the component will not **re-render**.

  - **componentDidMount()**

  This is invoked immediately after a component gets rendered and placed on the DOM. Now, you can do any DOM querying operations.

  - **render()**

  This method is defined in each and every component. It is responsible for returning a single root **HTML node** element. If you don't want to render anything, you can return a **null** or **false** value.

3. Updating Phase

- In this , get new **Props** and change **State**.
- This phase also allows to handle user interaction and provide communication with the components hierarchy.
- The main aim of this phase is to ensure that the component is displaying the latest version of itself.
- Unlike the Birth or Death phase, this phase repeats again and again. This phase consists of the following methods.

- **componentWillRecieveProps()**

It is invoked when a component receives new props. If you want to update the state in response to prop changes, you should compare this.props and nextProps to perform state transition by using **this.setState()** method.

- **shouldComponentUpdate()**

It is invoked when a component decides any changes/updation to the DOM. It allows you to control the component's behavior of updating itself. If this method returns true, the component will update. Otherwise, the component will skip the updating.

Updating Phase

- **componentWillUpdate()**

It is invoked just before the component updating occurs. Here, you can't change the component state by invoking **this.setState()** method.

- It will not be called, if **shouldComponentUpdate()** returns false.

- **render()**

It is invoked to examine **this.props** and **this.state** and return one of the following types: React elements, Arrays and fragments, Booleans or null, String and Number.

- If shouldComponentUpdate() returns false, the code inside render() will be invoked again to ensure that the component displays itself properly.

- **componentDidUpdate()**

It is invoked immediately after the component updating occurs.

- In this method, you can put any code inside this which you want to execute once the updating occurs. This method is not invoked for the initial render.

4. Unmounting Phase

- It is the final phase of the react component lifecycle.
- It is called when a component instance is **destroyed** and **unmounted** from the DOM. This phase contains only one method and is given below.
  - **componentWillUnmount()**

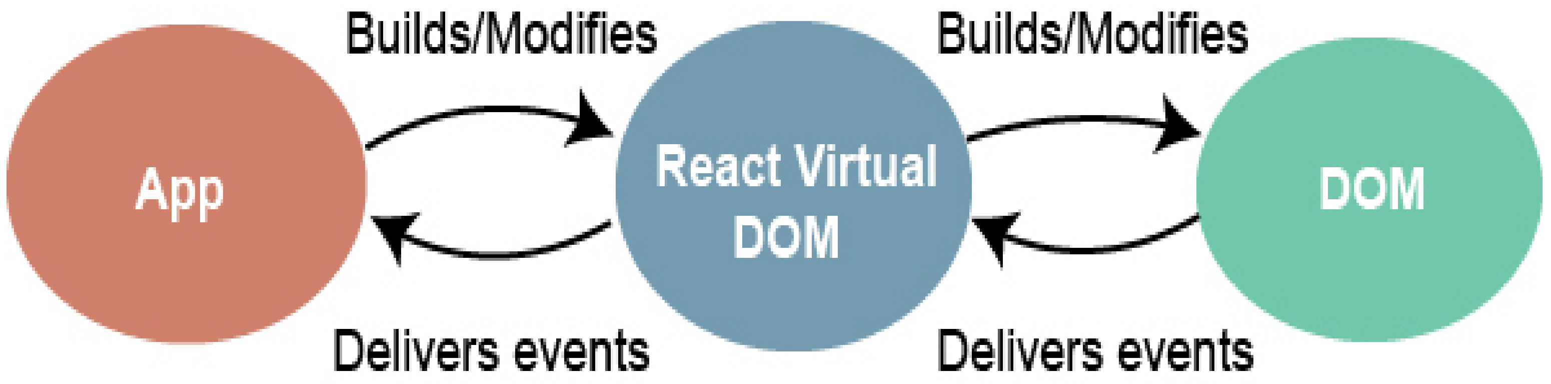
  This method is invoked immediately before a component is destroyed and unmounted permanently. It performs any necessary **cleanup** related task such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements.
- If a component instance is unmounted, you cannot mount it again.

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

Event handling



Handling events with react have some syntactic differences from handling events on DOM.

•React events are named as **camelCase** instead of **lowercase**.

•With JSX, a function is passed as the **event handler** instead of a **string**.

For example, the HTML:

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

is slightly different in React:

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

- Another difference is that cannot return false to prevent default behavior in React.
- **Must call preventDefault explicitly.**
- For example, with plain HTML, to prevent the default form behavior of submitting, we can write:

```
<form onsubmit="console.log('You clicked
   submit.'); return false">
  <button type="submit">Submit</button>
</form>
```

In React

```
function Form() {
  function handleSubmit(e) {
    e.preventDefault();
    console.log('You clicked submit.');
  }
  return (
    <form onSubmit={handleSubmit}>
      <button type="submit">Submit</button>
    </form>
  );
}
```

e is a synthetic event.

React Forms

- Forms are an integral part of any modern web application.
- It allows the users to interact with the application as well as gather information from the users.
- Forms can perform many tasks that depend on the nature of your business requirements and logic such as authentication of the user, adding user, searching, filtering, booking, ordering, etc.
- A form can contain text fields, buttons, checkbox, radio button, etc.

Creating Form

- React offers a stateful, reactive approach to build a form.
- The component rather than the DOM usually handles the React form.
- In React, the form is usually implemented by using controlled components.
- There are mainly two types of form input in React.
- Uncontrolled component
- Controlled component

CONTROLLED COMPONENT	UNCONTROLLED COMPONENT
Does not maintain its internal state.	Maintains its internal state.
Data is controlled by the parent component.	Data is controlled by the DOM itself.
Accepts its current value as a prop.	Uses a ref for their current values.
Allows validation control.	Does not allow validation control.
Better control over the form elements and data.	Limited control over the form elements and data.

Uncontrolled component

- The uncontrolled input is similar to the traditional HTML form inputs.
- The DOM itself handles the form data.
- Here, the HTML elements maintain their own state that will be updated when the input value changes.
- To write an uncontrolled component, need to use a ref to get form values from the DOM.
- In other words, there is no need to write an event handler for every state update.

- Can use a ref to access the input field value

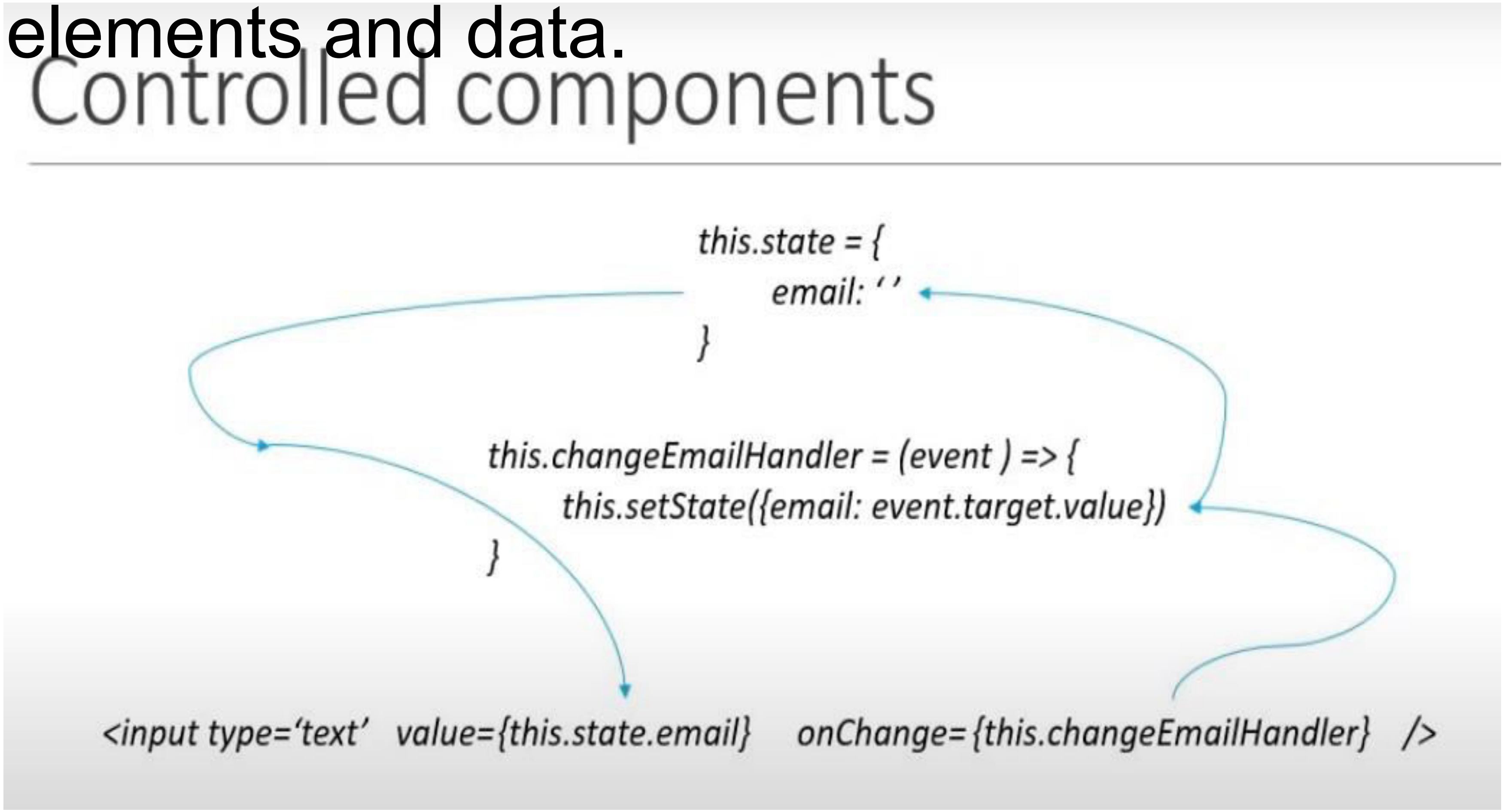
```
<form>
  <label>
    Name:
    <input type="text" name="name" />
  </label>
  <input type="submit" value="Submit" />
</form>
```

- This form has the default HTML form behavior of browsing to a new page when the user submits the form.
- If you want this behavior in React, it just works.
- But in most cases, it’s convenient to have a JavaScript function that handles the submission of the form and has access to the data that the user entered into the form.
- The standard way to achieve this is with a technique called “controlled components”.

Controlled Component

- In HTML, form elements typically maintain their own state and update it according to the user input.
- In the controlled component, the input form element is handled by the component rather than the DOM.
- Here, the mutable state is kept in the state property and will be updated only with **setState()** method.
- Controlled components have functions that govern the data passing into them on every **onChange event**, rather than grabbing

This data is then saved to state and updated with setState() method. This makes component have better control over the form elements and data.



Controlled components

```
this.state = {
        email: ''
}

this.changeEmailHandler = (event ) => {
    this.setState({email: event.target.value})
}

<input type='text'   value={this.state.email}    onChange={this.changeEmailHandler}   />
```

- A controlled component takes its current value through **props** and notifies the changes through **callbacks** like an onChange event.
- A parent component "controls" this changes by handling the callback and managing its own state and then passing the new values as props to the controlled component.
- It is also called as a "dumb component."

```
class NameForm extends React.Component
  {
 constructor(props) {
   super(props);
   this.state = {value: ''};
   this.handleChange =
  this.handleChange.bind(this);
   this.handleSubmit =
  this.handleSubmit.bind(this);
 }
 handleChange(event) {
   this.setState({value: event.target.value});
```

```
handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }
render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value}
   onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
```

The textarea Tag

- In HTML, a <textarea> element defines its text by its children:

<textarea> Hello there, this is some text in a text area </textarea>

In React, a <textarea> uses a value attribute instead.

```
class EssayForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 'Please write an essay about your favorite
     DOM element.'
    };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) {
    this.setState({value: event.target.value});
  }
```

```
handleSubmit(event) {
    alert('An essay was submitted: ' + this.state.value);
    event.preventDefault();
  }
render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Essay:
          <textarea value={this.state.value}
   onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
```

React -styling CSS

Inline Styling

```
import React from "react";
function App() {
return ( <h1 style={{ color: "red" }}> Hello
   World</h1> );
 }
export default App;
```

- Style attribute is an object, can also separate the style by writing it as a constant.

```
import React from "react";
const pStyle = {
  fontSize: '16px',
  color: 'blue'}
function App() {
  return (
     <p style={pStyle}>The weather is sunny
   with a small chance of rain today.</p>
  );}
```

import a .css file

JavaScript import syntax to import a .css file to your JavaScript file. You can then use the CSS class name in JSX elements that you want to style, like this:

Style.css

```
.paragraph-text {
  font-size: 16px;
  color: 'blue';
}
```

```
import React, { Component } from 'react';
import './style.css';
function App() {
  return (
        <p className="paragraph-text">
      The weather is sunny with a small
   chance of rain today.
     </p>
     );}
export default App;
```

React Animation

- The animation is a technique in which images are manipulated to appear as moving images. It is one of the most used technique to make an interactive web application.
- In React, we can add animation using an explicit group of components known as the **React Transition Group**.
- React Transition Group is an add-on component for managing component states and useful for defining **entering** and **exiting** transitions.
- It is not able to animate styles by itself.

- It makes the implementation of visual transitions much easier.
- React Transition group has mainly **two APIs** to create transitions. These are:
- **ReactTransitionGroup:** It uses as a low-level API for animation.
- **ReactCSSTransitionGroup:** It uses as a high-level API for implementing basic CSS transitions and animations.

Installation

- We need to install **react-transition-group** for creating animation in React Web application. use the below command.

- $ npm install react-transition-group --save

React Transition Group Components

- React Transition Group API provides **three** main components.
- Transition
- CSSTransition
- Transition Group

Transition

- It has a simple component API to describe a transition from one component state to another over time.
- It is mainly used to animate the **mounting** and **unmounting** of a component.
- It can also be used for in-place transition states as well.
- Can access the Transition component into four states:
- entering
- entered

CSSTransition

- The CSSTransition component uses CSS stylesheet classes to write the transition and create animations.
- It is inspired by the **ng-animate** library.
- It can also inherit all the props of the transition component.
- Divide the "CSSTransition" into **three** states.
- Appear
- Enter
- Exit

- CSSTransition component must be applied in a pair of class names to the child components.
- The first class is in the form of **name-stage** and
- the second class is in the **name-stage-active**.
- For example, you provide the name fade, and when it applies to the 'enter' stage, the two classes will be **fade-enter** and **fade-enter-active**.
- It may also take a prop as Timeout which

TransitionGroup

- This component is used to manage a set of transition components (Transition and CSSTransition) in a list.
- It is a state machine that controls the **mounting** and **unmounting** of components over time.
- The Transition component does not define any animation directly.
- Here, how 'list' item animates is based on the individual transition component.
- It means, the "TransitionGroup" component

What is React Router?

- React router is a library that allows you to handle routes in a web app, using dynamic routing.
- Dynamic routing takes place as the app is rendering on your machine, unlike the old routing architecture where the routing is handled in a configuration outside of a running app.
- React router implements a component-based approach to routing.
- It provides different routing components according to the needs of the application and

React Router

- Routing is a process in which a user is directed to different pages based on their action or request.
- ReactJS Router is mainly used for developing Single Page Web Applications.
- React Router is used to define multiple routes in the application.
- When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.
- React Router is a standard library system

- It provides the synchronous URL on the browser with data that will be displayed on the web page.
- It maintains the standard structure and behavior of the application and mainly used for developing single page web applications.

React Router Installation

- React contains three different packages for routing.
- **react-router:** It provides the core routing components and functions for the React Router applications.
- **react-router-native:** It is used for mobile applications.
- **react-router-dom:** It is used for web applications design.
- It is not possible to install react-router directly in your application.

- To use react routing, first, you need to install

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.
- Every router creates history object to keep track of the current location of the page.

- Example

Benefits of React Router:

- Add routing to different views/components on Single Page Applications
- Composable
- Easily add links after designing the webpage
- React Router conditionally renders certain components depending on the route from the URL.

React Best Practices

- React best practices, methods, and techniques that will help us stay consistent during the app development.
- **State** – The state should be avoided as much as possible. It is a good practice to centralize the state and pass it down the component tree as props. Whenever we have a group of components that need the same data, we should set a container element around them that will hold the state.
- Flux pattern is a nice way of handling the state in React apps.
- **PropTypes** – The PropTypes should always be defined. This will help is track all props in the app and it will also be useful for any developer working on the same project.

React Best Practices

- **Render** − Most of the app's logic should be moved inside the render method.
- We should try to minimize logic in component lifecycle methods and move that logic in the render method. The less state and props we use, the cleaner the code will be. We should always make the state as simple as possible. If we need to calculate something from the state or props, we can do it inside the render method.
- **Composition** − React team suggests to use a single responsibility principle. This means that one component should only be responsible for one functionality. If some of the components have more than one functionality, we should refactor and create a new component for every functionality.
- **Higher Order Components (HOC)** − Former React versions offered mixins for handling reusable functionalities. Since mixins are now deprecated, one of the solutions is to use HOC.

All the best!