React JS Notes:

Introduction:

- React is JavaScript library created by Facebook.
- Most popular JavaScript library for creating UI.
- Also used by Netflix & Instagram.
- Used to create Single Page Applications (SPA)
- We can build modern, fast Single Page Applications or websites with React.

Is React JS a Library or a Framework?

- React is a Library, not a Framework.

What is a Library?

- A library in programming can be explained as a collection of codes. We use a library to write code in a much simpler way or to import a feature from it into our project. JQuery is a library for example.
- We can write JavaScript much simpler by using JQuery, or we can import written JQuery features to our project. The project itself is not dependent on a library.

What is a Framework?

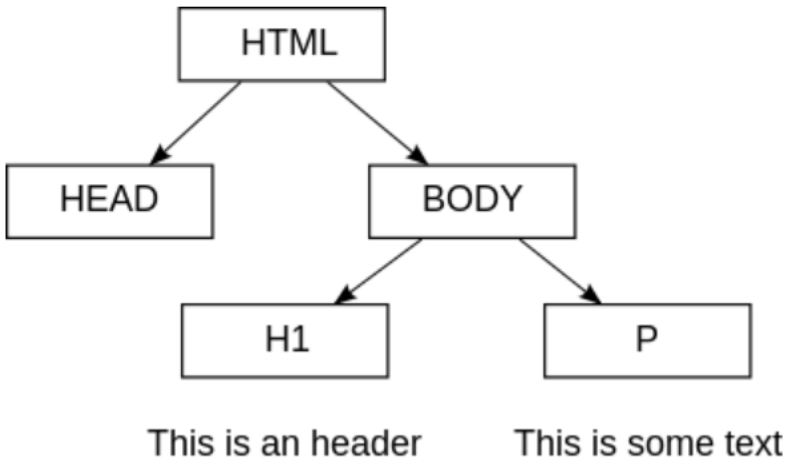
- A Framework, on the other hand, is a complete package of code with its own functionalities & libraries. A Framework has its own rules, you don't have much flexibility and the project is dependent on the Framework you use. Angular is an example of a framework.

How React actually works?

React Virtual DOM

- To understand the importance of React Virtual DOM, first, you need to know what DOM (Document Object Model) is.

- DOM is basically the representation of the HTML code in a webpage. The document is the webpage itself, the objects are the HTML tags. And finally, the model of DOM is a tree structure:



Tree Structure of the Document Object Model

What is the benefit of Virtual DOM?

- Each time you make a change in the code, DOM will be completely updated and rewritten. This is an expensive operation and consumes lots of time. In this point, React provides a solution: The Virtual DOM.
- So when something changes:
  - React first creates an exact copy of the DOM
  - Then React figures out which part is new and only updates that specific part in the Virtual DOM
  - Finally, React copies only the new parts of the Virtual DOM to the actual DOM, rather than completely rewriting it.
- This approach makes a webpage much faster than a standard webpage. That's also one of the reasons why React is so popular.

React's Core Syntax: JSX

- In classic Frontend programming, we have separated HTML, CSS and JS file structures. React is a bit different. We don't have separated HTML files in React.

- In JSX syntax, we write HTML tags inside JavaScript.
- In React, for example, a simple JavaScript variable can be like this: const element = <h1>Hello!</h1>;

- Normally, we can't assign an HTML tag to a JavaScript variable. But with JSX, we can. The code above you see is neither HTML nor JavaScript. It's an example of JSX.

So what is this JSX?

- JSX (JavaScript XML) is a syntax extension to JavaScript used by React. JSX is basically used to write HTML tags inside JavaScript. Later, the JSX code will be translated into normal JavaScript, by Babel.
- In summary, React doesn't have HTML files, HTML tags are rendered directly inside JavaScript. This approach makes React faster.

Do I have to work with JSX?

- You don't have to use JSX with React, but it is strongly recommended. JSX simplifies React and makes it easier to read. Let me give an example of React code with and without JSX.

React with JSX:

```
class Hello extends React.Component {
  render() {
    return <div>Hello {this.props.toWhat}</div>;
  }
}

ReactDOM.render(
  <Hello toWhat="World" />,
  document.getElementById('root')
);
```

React without JSX:

```
class Hello extends React.Component {
  render() {
    return React.createElement('div', null, `Hello
${this.props.toWhat}`);
  }
}

ReactDOM.render(
  React.createElement(Hello, {toWhat: 'World'}, null),
  document.getElementById('root')
);
```

Some important rules about JSX:

- We can't return more than one HTML element at once, but we can wrap the elements inside a parent HTML tag:

```
class Test extends React.Component {
  render() {
    return (
      <div>
        <p>Hello</p>
        <p>World</p>
      </div>
    );
  }
}
```

- We can use JSX inside for loops, if-else cases:

```
render() {
    if(condition==true) {
        return <p>This text</p>;
    } else {
        return <p>Another text</p>;
    }
}
```

- HTML attribute names like "class" becomes "className".

```
<div className="myClass"></div>
```

- HTML tags must always be closed.

React Installation:

- React requires Nodejs.
- After installing Nodejs, open your Terminal or Command Prompt and type the following command to create your React app:

```
npx create-react-app my-app

cd my-app
```

- npx - It is a tool for executing node packages. [node package runner.] x - executor.

What is a React Component?

- A component is an independent, reusable code block, which divides the UI into smaller pieces.
- In other words, we can think of components as LEGO blocks. Likewise we create a LEGO structure from many little LEGO blocks, we create a webpage or UI from many little code blocks (components).
- Smaller code blocks. Easy to maintain. They are reusable, easier to read, write & test.
- React has 2 types of components: Functional (Stateless) and Class (Stateful).

Functional (Stateless) Components:

- A functional component is basically a JavaScript (or ES6) function which returns a React element. According to React official docs, the function below is a valid React component:

```
function Welcome(props) {
  return <h1>Hello, {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. — reactjs.org
- So we can define a React functional component as a JS Function:

```
function Example() {
  return ( <h1>I'm a functional component!</h1> );
};
```

- or as an ES6 arrow function:

```
const Example = () => {
  return ( <h1>I'm a functional component!</h1> );
};
```

- Both of the functions are valid React components. They may take props as an argument (when necessary), but they must return a React element.
- IMPORTANT: Functional components are also known as stateless components because, in the past, we couldn’t do more complex things like React State (data) management or life-cycle methods in functional components.
- However, React introduced React Hooks in version 16.8, which now allows us to use state & other features in functional components.
- So a React Functional Component:
  - is a JavaScript / ES6 function
  - must return a React element
  - take props as parameter if necessary

Class (Stateful) Components

- Class components are ES6 classes. They are more complex than functional components including constructors, life-cycle methods, render( ) function and state (data) management.

- In the example below, we can see how a simple class component looks like:

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

class ExampleComponent extends Component {
  render() {
    return (
      <div>This is an example component.</div>
    );
  }
}

export default ExampleComponent;
```

- Here, the ExampleComponent class extends Component, so React understands that this class is a component, and it renders (returns) a React Element.
- So, a React class component:
    - is an ES6 class, will be a component once it 'extends' React component.
    - can accept props (in the constructor) if needed
    - can maintain its own data with state
    - must have a render( ) method which returns a React element (JSX), or null

How to call a component?

- A component is being called like an HTML tag, but starting with a capital letter:

<ExampleComponent />

- Components are the core of React. Having a better knowledge of when and how to use functional & class components not only makes your React app better performance, readable and testable, but also makes you a better programmer.

What is “Props” and how to use it in React?

- React has a different approach to data flow & manipulation than other frameworks.

What is Props?

- React is a component-based library which divides the UI into little reusable pieces. In some cases, those components need to communicate (send data to each other) and the way to pass data between components is by using props.
- “Props” is a special keyword in React, which stands for properties and is being used for passing data from one component to another.
- But the important part here is that data with props are being passed in a uni-directional flow. (one way from parent to child)
- Furthermore, props data is read-only, which means that data coming from the parent should not be changed by child components.

Using Props in React

- Firstly, define an attribute and its value(data)
- Then pass it to child component(s) by using Props
- Finally, render the Props Data

```
class ParentComponent extends Component {
  render() {
    return (
      <h1>
        I'm the parent component.
        <ChildComponent />
      </h1>
    );
  }
}
```

```
const ChildComponent = () => {
  return <p>I'm the 1st child!</p>;
};
```

- The problem here is that, when we call the ChildComponent multiple times:

```
class ParentComponent extends Component {
  render() {
    return (
      <h1>
        I'm the parent component.
        <ChildComponent />
        <ChildComponent />
        <ChildComponent />
      </h1>
    );
  }
}
```

- It always renders the same string again and again:

- But what we like to do here is to get dynamic outputs, because each child component may have different data and let's see how we can solve this issue by using props…
- 1st Step: Defining Attribute & Data
- We already know that we can assign attributes and values to HTML tags:

```
<a href="www.google.com">Click here to visit Google</a>;
```

- Likewise, we can do the same for React components. We can define our own attributes & assign values with interpolation { }:

```
<ChildComponent someAttribute={value} anotherAttribute={value}/>
```

- Let's declare a "text" attribute to the ChildComponent and then assign a string value: "I'm the 1st child".

```
<ChildComponent text={"I'm the 1st child"} />
```

- Now the ChildComponent has a property and a value. Next, we need to pass it via Props.
- 2nd Step: Passing Data using Props
- OK, now let's take the "I'm the 1st child!" string and pass it by using props.
- Passing props is very simple. Like we pass arguments to a function, we pass props into a React component and props bring all the necessary data.
- Arguments passed to a function:

```
const addition = (firstNum, secondNum) => {
  return firstNum + secondNum;
};
```

- Arguments passed to a React component:

```
const ChildComponent = (props) => {
  return <p>I'm the 1st child!</p>;
};
```

- Props are arguments passed into React components.
- Final Step: Rendering Props Data
- Prop is an Object
- we will render the props object by using string interpolation:
- {props}
- Log props to the console.
- console.log(props);

```
▶ Object { text: "I'm the 1st child" }
```

- As we can see, Props returns back an object. In JavaScript, we can access to object elements with dot(.) notation. So, let's render our text property with interpolation:

```
const ChildComponent = (props) => {
  return <p>{props.text}</p>;
};
```

- And that's it! We've achieved to render the data coming from the parent component.
- let's do the same for other child components:

```
class ParentComponent extends Component {
  render() {
    return (
      <h1>
        I'm the parent component.
        <ChildComponent text={"I'm the 1st child"} />
        <ChildComponent text={"I'm the 2nd child"} />
        <ChildComponent text={"I'm the 3rd child"} />
      </h1>
    );
  }
}
```