Week1

-JSX is an XML/HTML-like syntax used by React that extends ECMAScript so that XML/HTML-like text can co-exist with JavaScript/React code.

-React is all about components.

-The createElement function is the main function in the React top-level API.

-You can use JavaScript expressions anywhere in JSX.

-JSX stands for JavaScript XML.

-You can write React components with JavaScript classes.

-All React elements attributes (events included) are named using camelCase.

-An actual JavaScript function reference is passed as the event handler, rather than a string.

-Every React component has a story.

-React components can have a/an private state.

-React get its name from the fact that it reacts to state changes.

-We can think of React as the agent we hired to communicate with the browser.

-A component might need to re-render whet its stat get updated or when its parent decides to change the props that it passed to the component.

-Lifecycle methods are actually escape hatches.

-If you’re not doing anything special, you can create full applications without lifecycle methods.

-React lets you compose complex User Interfaces (UI) from small and isolated pieces of code called components.

-When our data changes, React will efficiently update and re-render our components.

-The render method returns a description of what you want to see on the screen.

-JSX comes with the full power of JavaScript.

-Each React component is encapsulated and can operate independently.

Week2

* Props are read-only.
* To add just a few style properties, inline styling is the best option.
* In React, you usually render lists inside a component.
* A component with a/an render prop takes a function that returns a React element and calls it instead of implementing its own render logic.
* To reuse style properties in the same file, then style-components are optimal.
* A key is a special string attribute you need to include when creating lists of elements.
* All React components must act like pure functions with respect to their props.
* In React, inline styles are specified with an object whose key is the camelCased version of the style name, and whose value is the style’s value, usually a string
* The best way to pick a key is to use a string that uniquely identifies a list item among its siblings.
* A render prop is a function prop that a component uses to know what to render.
* Styled-components is a library for React and React Native that allows you to use component-level styles in your application that are written with a mixture of JavaScript and CSS.
* All you need to render a list in React is Array.map.
* You can implement most higher-order components (HOC) using a regular component with a render prop.
* CSS classes are generally better for performance than inline styles.
* It’s been said that most web apps are just lists of things.
* The term render prop refers to a technique for sharing code between React components using a prop whose value is a function.
* Inline styles are used when a single HTML element needs unique styles.
* In React, when you use lists, each list item needs a unique key.
* Any prop that is a function that a component uses to know what to render is technically a “render prop”.
* To style and element with an inline style attribute, the value must be a JavaScript object.

Week3

* Class components can define functions that will execute during the component’s lifecycle.
* Class syntax is one of the most common ways to define a React component.
* To create a class component, just define a class that extends Component and has a render function.
* One of the benefits class components have over functional components is access to component state.
* Class components are stateful components.
* Stateful components keep track of changing data, while stateless components print out what is given to them via props.
* All React components must act like pure functions with respect to their props.
* Regardless of which modern framework you use, Angular, Vue, or React, they are all component-based.
* Component-based development is today’s popular way for building User Interfaces and Web applications.
* Class components are ECMAScript 6 (ES6) classes.
* Class components are more complex than functional components.
* A class components will be a component once it extends React component.
* Class components can accept props (in the constructor) if needed.
* Class components can maintain their own data with state.
* A class component must have a render( ) method which returns a React element (JSX), or null.
* A class component is called like an HTML tag, but starts with a capital letter.
* Having a better knowledge of when and how to use functional and class components makes React applications perform better, more readable, and more testable.
* The simplest way to define a component in React is to write a JavaScript function.
* The most obvious difference between a functional component and a class component is syntax.
* A feature you cannot use in functional components versus class components are lifecycle hooks.

Week4

* React component lifecycle has three categories: Mounting, Updating, and Unmounting.
* The render() is the most used lifecycle method.
* As your app grows, you can catch a lot of bugs with type checking.
* Flow is a static type checker for your JavaScript code.
* There are two types of form input in React: uncontrolled input and controlled input.
* In React, the uncontrolled form input is like traditional HTML forms input, they remember what you typed.
* Commonly used lifecycle methods include render() and constructor().
* In React, after you’ve rendered your component for the first time, componentDidMount is called.
* Static type checkers like Flow and TypeScript identify certain types of problems before you even run your code.
* Flow does not require the react preset, but they are often used together. Flow itself understands JSX syntax out of the box.
* In most situations, we will have more than one form input.
* Just like in HTML, React uses forms to allow users to interact with the web page.
* In HTML, form data is usually handled by the DOM. In React, form data is usually handled by the components.
* With forms, you must initialize the state in the constructor method before you can use it.
* In the Mounting lifecycle phase, Mounting means putting elements into the DOM.
* The render() method is required and will always get called in the Mounting lifecycle phase.
* Just like in HTML, React uses forms to allow users to interact with the web page.
* A component is updated whenever there is a change in the component’s state or props.

Week5

* Components are the heart of React's powerful, declarative programming model. React Router is a collection of navigational components that compose declaratively with your application.
* The react-dom package serves as the entry point to the DOM and server renderers for React.
* Whether you want to have bookmarkable URLs for your web app or a composable way to navigate in React Native, React Router works wherever React is rendering.
* When building a web app, you can use react-router-dom to install React Router.
* If you use ES6 with npm, you can write “import ReactDOM from 'react-dom'”.
* React Router is the standard routing library for React.
* The react-dom package provides DOM-specific methods that can be used at the top level of your app and as an escape hatch to get outside of the React model if you need to.
* React Router keeps your UI in sync with the URL.
* FindDOMNode is an escape hatch used to access the underlying DOM node
* When a route has one or more named components, the child elements are available by name on this.props.
* FindDOMNode only works on mounted components (that is, components that have been placed in the DOM).
* An essential part of many applications is the ability to read route parameters from a URL.
* Portals provide a way to render children into a DOM node that exists outside the hierarchy of the DOM component.
* You can pass in query strings as props to any component that will be rendered at a specific route, and access these parameters as props.location.query.
* In the Mounting lifecycle phase, Mounting means putting elements into the DOM.
* React Router, and dynamic, client-side routing, allows us to build a single-page web application with navigation without the page refreshing as the user navigates.

Week6

* Context provides a way to pass data through the component tree without having to pass props down manually at every level.
* Context is designed to share data that can be considered global for a tree of React components, such as the current authenticated user, theme, or preferred language.
* Using context, we can avoid passing props through intermediate elements.
* Context is primarily used when some data needs to be accessible by many components at different nesting levels.
* Applying context should be used sparingly because it makes component reuse more difficult.
* If you only want to avoid passing some props through many levels, component composition is often a simpler solution than context.
* Sometimes, the same data needs to be accessible by many components in a tree, and at different nesting levels. Context lets you broadcast such data, and changes to it, to all components below.
* React.createContext() creates a Context object.
* Every Context object comes with a Provider React component that allows consuming components to subscribe to context changes.
* The contextType property on a class can be assigned a Context object created by React.createContext().
* Context.Consumer is a React component that subscribes to context changes.
* Context can let you build an API where parents and children communicate.
* React Context allows us to define data stores and access them where they are needed - we don’t have to pass down data through properties.
* With React Context we can define something like an “application global state” and use that data where needed.
* React Context API is a way to essentially create global variables that can be passed around in a React app.
* Context is often touted as a simpler, lighter solution to using Redux for state management.
* The context provider always needs to exist as a wrapper around the parent element, no matter how you choose to consume the values.
* The way you provide Context is the same for class and functional components, but consuming it is a little different for both.
* The Context API is a component structure provided by the React framework, which enables us to share specific forms of data across all levels of the application.