React

By facebook

1.JSX

- Uses a syntax extension for JS called JSX which allows you to write html directly into JS.
- As JSX is an extension you can write JS in JSX just need to encapsulate it inside { brackets
- Because JSX is not valid JS , JSX code must be compiled into Js code Babel is popular tool for this process.
- Nested JSX will always return a single element.

*. **{“**This is treated as JS code**”}**

Under the hood the FCC react challenges are calling
ReactDOM.render(JSX,document.getElementById(‘root’))
But unlike the JFX reacts components will be like this
ReactDOM.render(<ComponentToRender />,targetNode)

Commenting JSX code is like this -> **{/* */}**

How is JSX different from HTML

1. You can no longer use the word class to define classes,because class is a reserved work in JS instead we use **className** in JSX
2. Any JSX element can be written with a self-closing tag and every element must be closed. Like
 should be

1.2 Rendering into DOM

- We can render this JSX directly into HTML DOM using React’s rendering API known as ReactDOM

ReactDOM.render(componentToRender, targetNode)

2.React

A typical react component is an ES6 class which extends react.Component

Components are the core to react.

1.To create a stateless component, you simply write a **JS function** that **returns either JSX or null.**

- React requires you **function_name *to being with capital letter***

2.The other way to define a react component is with ES6 class syntax.

```
Class Kitten extends React.Component{
constructor(props){
    super(props);}
render(){
return(
<h1>hi</h1>);
}}
```

If you have three diff components like navbar , dashboard and footer.first create a parent component called app which renders these 3 component for which you will write the render() like this

```
return(
    <App>
    <navbar />
    <Dashboard />
    <Footer />
    </App>
)
```

2.1 React's Props

In React, you can pass **props(properties)** to child components.

- Ex. You have an **app** which renders a child component called **welcome** which is a stateless functional component. You can pass welcome a user prop by writing

```
<App>
    <Welcome user=”Mark” />
</App>
```

You can use custom attributes created by you and supported by react to be passed to component.

Since Welcome is a stateless functional component, it has access to this value like this.

```
Const Welcome =(props) => <h1> Hello,
{props.user}!</h1>
```

It is standard to call this value prop. And when dealing with stateless functional components , you basically consider it as an argument to a function which return JSX.

You can access the value of the arguments i.e props within the function body

React also has an option to set default props.This means you can specify what the prop value should be if no valued in provided

```
MyComponent.defaultProps={location: ‘san
Francisco’}
```

To override a default prop value the syntax should be

```
<Component propName={value}/>
```

You can set the propTypes on your component to require the data to be of type(type you want string or array),It is considered best

practice to set propTypes when you know the type of the prop ahead of time.

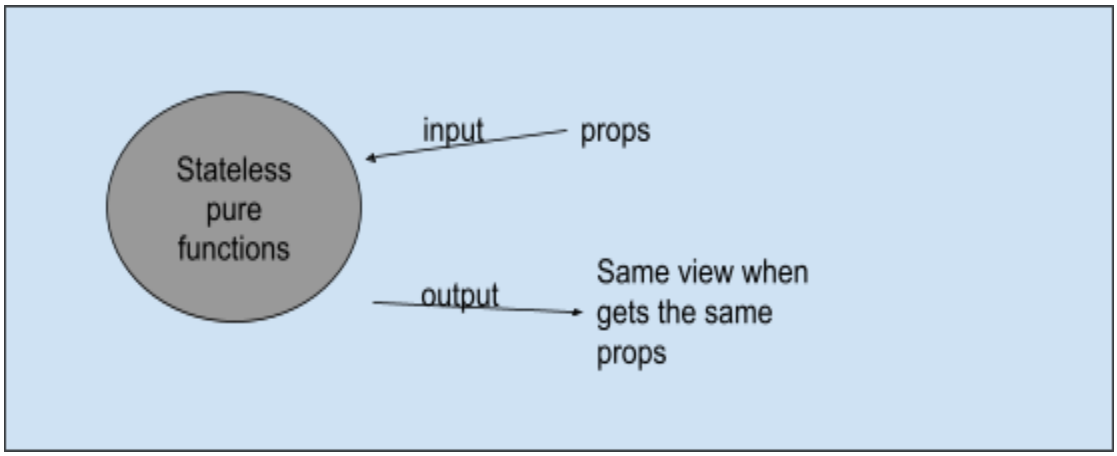
```
Mycomponent.propTypes= {handleClick:
PropTypes.func.isRequired}
```

In the example above, the PropTypes.func part checks that handleClick is a function. Adding isRequired tells React that handleClick is a required property for that component. You will see a warning if that prop isn't provided. Also notice that **func** represents function.

Among the seven JavaScript primitive types, **function** and **boolean** (written as **bool**) are the only two that use unusual spelling.

In ES6 the way to access a prop is using **this** keyword like:X

```
In the parent set <component name=”some”/>
In component
<div>
    <p>Hello
,<strong>{this.props.name}</strong></p>
</div>
```



- A **stateless function component** is any function you write which accepts props and returns JSX.
- A **stateless component** on the other hand is a class that extends React.component.
- A **Stateful component** is a class component that does maintain its own internal state.stateful component are referred as react components

A common practise is to minimize statefulness and to create stateless functional components whenever possible

2.2 React's State

State consists of any data you application needs to know about, that can change overtime,
You want your app to respond to state change and present an updated UI when necessary.

Declaring a state
You create a state property on the component class in its constructor.This initialized the component with state when its created

```
constructor(props){
    super(props)
    this.state={ }
            }
```

You have access to the state through out your component's lifecycle
You can **update, pass it a prop to child component** , render it in UI
Accessing the state in the render() you have to enclose it inside {}
`{this.state.statePropertry}`

There is another way

1. In the render() method , before the return statement you can write JS code directly. For example- You can access data from state or props , perform computations on this data and so on. Then you can assign any data to variable , which you have access to in the return statement.
2. There is also a way to change the component's state. React provides a method called **setState**

   ```
   this.setState({
       Key: "value"
   )
   ```

   React expects you to never modify state instead always use this.setState() when state change occurs

2.3 React's class method

You can also define methods for your component class. A class method typically need to use **this keyword** so it can access properties on the class (state and props) inside the scope of the method

Few ways to allow your class methods to access **this**

- Common way is to explicitly bind this in the constructor so this becomes bound to the class method when component is intializes.
- `You may have noticed the last challenge used` **`this.handleClick = this.handleClick.bind(this)`** `for its handleClick method in the constructor. Then, when you call a function like this.setState() within your class method, this refers to the`

```
    class and will not be undefined.in the
    render() we have onClick={this.handleClick}
```

Sometimes you might need to know the previous state when updating the state.However state updates may be asynchronous- this means React may batch multiple setState() calls into a single update.

Wrong Approach -
```
this.setState({
                Counter:
this.state.counter+this.props.increment });
```

Right Approach-
```
this.setState((state,props)=>({
    Counter: state.counter+props.increment }));
```

RA without Props -
```
this.setState(state => ({
    Counter: state.counter+1 }));
```

You have to wrap the object literal in parentheses otherwise JS thinks its a block of code

Something that bothering me in the FCC react challenge

```
toggleVisibility(){
    this.setState(state=>{
        if(state.visibility===true{
            Return {visibility: false};}
        Else {
            Return {visibility: true};}
    )}
```

Here I don't see the (after =>

And yet the code seems to work just Fine how? And why

Your application may have complex interaction between state and rendered UI.like form control elements such as elements for text-input like input and textarea maintain their own state in the DOM *with react you can move this mutable state into the React component's state.The user's input become part of the application state.*

In react, the data flow is unidirectional
Complex stateful apps can be broken down into just a few or maybe a single stateful component.

How State Works in React- Explained with Code example

(link)-*https://www.freecodecamp.org/news/what-is-state-in-react-explained-with-examples/*

In react when binding a state to a function we use bind
But this documentation tell us to use this instead

```
handleClick = () => {
  this.setState((prevState) => {
    return {
      counter: prevState.counter + 1
    };
  });
};
```

As arrow functions do not have their own this context, it will take the context as the class so there is no need to use the .bind method.

However i have been using the this with the above format minus arrow function and i still had to use .this bind with function call may be its an issue with the missing arrow function syntax.

Introduction to react Hooks by Yogesh chavan

Before react hooks , there was no way of using state and lifecycle methods in functional components thats why they were called stateless functional components.
To use react hooks, we just need the latest version of create-react-app

```
Npm install react@latest react-dom@latest
```

Hooks are functions that let you "hook into" the react state and lifecycle features in functional components

Some important built-in hooks are

1. **UseState hook**

Parameter- initial value of state
Returns-array whose first value =current value of state & second value= function which we will use to update the state similar to setState method

In class based components, state is always an object but in useState we can use any value like num,string,book,obj,arr,null etc.

Note-

- When using object in usestate ,state is not merged when updating the state.
- Using an object in usestate will completely replace the object instead of merging it when we set state using setState method.

- That's why its recommended to use multiple *useState* calls instead of single useState to store object.
- **If you still want to use a single useState then you need to merge the previous state manually.**

2. **UseEffect hook**
   Using this hook we can implement lifecycle methods in functional components.lets you perform side effects in a functional component.

   Parameter-function & optional array as second argument

   Data fetching, setting up a subscription and manually changing the DOM in React components are all examples of side effects

   UseEffect server the same purpose as **componentDidMount, componentDidUpdate & componentWillUnmount** combined together.

```
useEffect(() => {
console.log("This will be executed when only
counter is changed");
}, [counter]);
```

So by providing the variable in dependencies array, the effect will be executed only when those variables are changed.

If you have multiples dependencies , you can specify that as a comma separated value in dependencies array.

```
useEffect(() => {
console.log("This will be executed when any of the dependency is changed");
}, [counter, some_other_value]);
```

visit An Introduction to React Hooks. Understand the built-in hooks and… | by Yogesh Chavan | Level Up Coding to learn about cleaning up the effect

3. useRef Hook

This hook allows us to access DOM element which is same as working with ref in React.

```
Syntax- const refContainer=useRef(initialValue);
```

Where initial value is optional

Returns a mutable ref object whose .current property is initialized to the passed argument (initialvalue)

4. useReducer Hook

This hook allows us to use redux for handling complex calculations without the need to installing the redux library

```
Syntax- const [state,dispatch]=useReducer(reducer, initialState);
```

`Returns currentstate and a dispatch function` which we can use to dispatch the actions

Creating Custom Hooks for later