(26) State in React → Declarative vs Imperative Programming.

→ User Interface = f(state)
[i.e a function of state].

→ State changes with changing needs with user interface.
& vice versa.

→ ∴ If state goes into true state user interface will be diff.
If state goes into false state user " " "] As we saw before in rendering

→ Declaritive ⇒ UI dependent on state.
Imperitive ⇒ what we have been doing. ∴ we tell the UI to do exactly what we want, not change with state.

Eg :- Case - 1
User Login
→ we already see if user is logged in (or) not & show either Hello (or) login screen
↓
Based on state (True (or) false)

Case - 2
we ask user if he is already logged in (or) not & based on his reply, give him the site.
↓
Target specific element

→ This is where the concept of hooks comes in, the most commonly used hook is useState & we are going to use that.

React Hooks - UseState:

→ 0
⊕ → (when we press ⊕ value → +1. everytime) using Onclick. } what we are making i.e to do something when button is clicked.

→ For that to happen we have to re-render our React DOM.

1st way → Place React.DOM inside the fn & in usual place ∴ making it re-render.

→ But it is inefficent.

2nd way → Use the hooks.

→ hooks can only be used in functional components.

Eg:-

→ function App() {
const state1 = React.useState(); → ①
(initial value, fn).
↳ Not needed if you import {useState}.

→ To call the states → State[0] can be used, but to make our code better we will call the state after destructuring the array.

↳ const rgb = [9, 132, 227]
then call → rgb[0], rgb[1], etc..
(OR)
const [red, green, blue] = [9, 132, 227]
then call → blue, green, etc..

∴

→ ① becomes const [count, changeCount] = React.UseState(34);

& function dosomething() {
changeCount(what to do with count, Eg:- count + 1);

can we more than one for diff buttons doing diff thing with count.?

(29) Javascript ES6 Object and Array Destructuring.

→ A Good practice to console.log after importing to see if & how it imported.

→ const [cat1, dog1] = animals;
  └ imported.

(OR)

is same as → var cat1 = animals[0];
             var dog1 = animals[1];

} Cat 1, dog 1 can be named anything

→ You can also destructure obj (like cat here)

Eg:- → const { name*1, sound1 } = cat;
       console.log(sound1);

} *name & sound here to match the properties in animal array.

(OR)

→ Equivalent of animal[0].name*

You can customize obj naming i.e → Used when APIs var names don't make sense

const { name: catName, sound: catSound } = cat;
console.log(catSound);

sound → Name used in dataset.
catSound → Name you want

Also

const { name = "Fluffy", sound = "Purr" } = cat;

→ will print this as default if nothing is present in dataset in it's place.

Used when API names (data) have blank components but we need a value.

For nested components →
const { name, sound, feeding: { food, water } = cat;
console.log(food).

31) Event Handling in React.

→ Event handling means changing the state of an obj.

→ An event can be a click, hover (or) any thing else.

→ So the things that will change (states) when an event occurs should be written in hooks.

→ ∴ By using hooks and html elements/Attributes like onClick we can handle events.

[Good challenge in this video also].

32) React forms

→ we will always be creating forms → for registration, for contacts (or) for signing in.

→ let's see handling forms then.

→ Like button → Onclick
Input → Onchange (i.e when something is typed)

→ Can get value, placeholder & type in console.log.
value → 'Ab', placeholder → what we had put., type → (text)

★ Can study controlled component from React documentation. (Should)

→ 2 change things onClick you can use 2 const (usestates vars).
(Just an example).

→ <form> component in html implements code & quickly refreshes the </form> form, this is it's default behaviour.

↳ Can use it, but use event.preventDefault(); to stop from refreshing.

(33) Class Components vs Functional Components.

→ Hooks vs classes and why classes are being replaced by hooks.

→ Hooks introduced in 2018. [so func comp directly can be used ~~never~~ need to covert in to class comp.]

→ Cleaner & less repetitive code. Easier to manage state.

→ Class & [redacted] hooks can be used side by side (if absolutely necessary).

(Synthetic Events in documentation)

(34) Changing Complex State.

→ What if we have many states at once. we have only discussed one change at a time.

→ Explained using concept of forms.

General procedure

→ function App () {

const [fName, setFName] = useState(" ");

(event passed as parameter or) Use setFName in fn to change it (Like a onChange, OnClick fn. Onchange = {fn})

→ Use fName in (return → html).

→ If we are using more than one then either we declare this again i.e have 2 (or) more const & set fns & then use them.

(OR)

Use const [fName, setFName] = UseState(" fName ", "lName");
& only have one fn.
↳ but inside this another fn (or) if & else should be used to handle diff var.

(35) Javascript ES6 Spread Operator.

→ Used to reduce & simpilify code.

→ ... → spread operator. } combining 2 arrays.

const fruits ["Apple", "Banana", ...vegetables];

↙ can be at any position.

↳ vegetable array appended.

→ This operator (similarly) can be used for objects as well.

Extra points not related: To use arrayname declared inside other fn in arrow fn we use

[keyname]: valueof.key.

---

(36) Manging a Component Tree.