||Hare Krishna||

Miracle
Pg.No :
Date: / /20

React

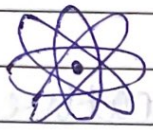
* Create a HTML page (index.js)

```
import React from "react"
import ReactDOM from "react-dom"

const page = (                    → JSX should always be nested under a single parent element
    <div>                                          style
        <img src="./__.png" width="40px" />        add closing tag in a
        <h1> Fun facts ... </h1>
        <ul>                                       children elements
            <li> ___ </li>
            <li> ___ </li>
            .
            .
            .
        </ul>                     → Same as HTML
    </div>
)

ReactDOM.render(page, document.getElementById("root"))
   su render karwa de        kya aagad render karwa de
```

→ index.html

```
<html>
  .
  .
  .
  <body>
      <div id="root"></div>
      <script src="index.pack.js"></script>
  </body>
</html>
```

→ Output : (index.html)

 → image

Fun Facts..... → h1

- :
- :
- :
- :

will

✱ React is composable and declarative.

✱ Quiz :

1) Why do we need to 'import React from "react"' in our files?

→ React is what defines JSX.

2) If I were to console.log(page) in index.js, what would show up?

→ A javascript object. React elements that describe what React should eventually add to the real DOM for us.

3) What's wrong with this code:

```
const page = (
    <h1> ___ </h1>
    <p> ___ </p>
)
```

→ We need our JSX to be nested under a single parent element.

4) What does declarative and imperative means?

→ Declarative : I can tell the computer WHAT to do and expect it to handle the details.

Imperative : I need to tell it HOW to do each step. (Like in JS)

5) What does 'composable' means?

→ Small pieces we put together to make something larger / greater than the individual pices.

✱ Custom components

→ PascalCase : First letter of each word is capital.

→ camelCase : First letter small, else capital.

→ Components has PascalCase convention.

→ Custom components : Independent pieces of functionality that you can reuse in your application, and are the building blocks of all React applications.

→ They are simple JavaScript functions and classes, but you can use them as if they were customized HTML elements.

→ Function component :

e.g.
```
function Page() {
  return(
    <>
      html code
    </>
  )}
```

single parent element ←

This is called a component

JSX

```
ReactDOM.render(<Page />, doc.getElementId(" "))
```

★ Quiz

1) What is a react component ?

→ A function that returns React elements (UI)

2) What's wrong in the code ?

```
function myComponent(){
  return(
    <small>___ </small>
```

```
    )
}
```

→ Pascal Case in function name (component name) i.e. MyComponent()

3) What's wrong with the code?

```
function Headr() {
    return (
        <header>
        :
        </header>
    )
}

ReactDOM.render(Headr(), doc.get... (""))
```

→ Should be like HTML tag in components i.e. < Headr />

✱ Parent / Child Components

e.g

```
function Headr() {
    return (
    )
}

function Footr() {
    return (
```

```
    )
}
```

→ Parent component

```
function Page() {
    return(
        <div>
            <Header />  } Child components
            <Footer />  }
        </div>
    )
}
```

ReactDOM.render(<Page />, doc...)

✱ Styling with classes

e.g. <ul className="nav-items">
:

→ in style.css file
.nav-item {
}

Instead of class, we use className in react.

✱ Organize components

→ import and export.

→ Create new files for each component.

e.g. Header.js → import React from "react" in all files where we use JSX

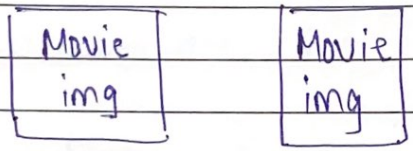
export default function Header() { ... }

→ index.js

import Header from './Header'

✱ React is data-driven

e.g. Netflix page



Mostly in all websites like Amazon, Netflix this type of component keeps getting repeated. And the data also is not hard-coded.

The data is stored in database and we just have to add img, name etc to it. So, if we do it statically we have to keep creating components again and again.

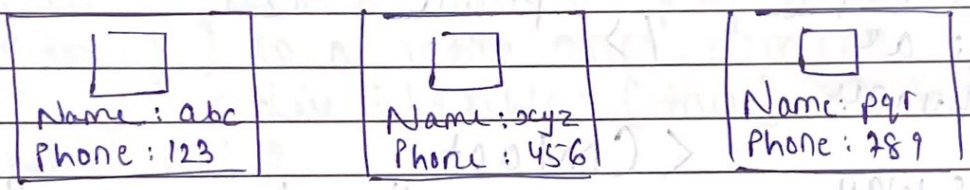
So, it can be said that it is not reusable. Here, comes the PROPS.

So, we can create a component and pass some parameter. In this way it can be reusable.

Props:

Props are arguments passed into React components.

e.g. Output:



Code (using props)

→ contact.js

argument

```
export default function Contact(props) {

  return (
    <div>
      <img src={props.img} />
      <h3>{props.name}</h3>
      <p>{props.phone}</p>
    </div>
  )
}
```

→ App.js

```
import Contact from "./Contact"
```

⋮

```
export default function App() {
    return (
        <div>
            <Contact
                img = "./images/abc.png"
                name = "Abc"
                phone = "123"
            />
            <Contact
                img = ".../xyz.png"
                name = "xyz"
                phone = "456"
            />
            <Contact
                img = "   "
                name = "pqr"
                phone = "789"
            />
        </div>
    )
}
```

props. — je name aapiyu hoi ae lakhvanu.

In this way, we can use the same 1 function for 3 different arguments. Same is done in Netflix, Amazon, Youtube for reusable components

✱ Prop quiz

1) What do props help us accomplish?

→ Props passed into a ~~function~~ component helps us make a component more usable

2) How do you pass a prop into a component?

→ <MyHeader title="Khushi" />
Component prop

3) Can I pass a custom prop (e.g. 'blahbla={true}') to a native DOM element (e.g.) Why or why not?

→ No.
Because the JSX we use to discribe the native DOM elements will be turned into Real DOM elements by React. And real DOM elements only have the property / attributes specified in the HTML specification (which doesn't include properties like 'blahbla')

4) How do I receive props in a component?

→ function Navbar (props) {
return (
:
)}

5) What datatype is 'props' when the component receives it?

→ An object!

→ Another way of writing props.

```
function Contact({img, name, phone}){
    return(
        <div>
            <img src={img}>
            <h1>{name}</h1>
            <p>{phone}</p>
        </div>
```

directly into func. but name should be same as defined whil passing prop

```
<Contact
    img=
    name=
    phone=
/>
```

✳ Array.map()

map() method creates a new array populated with the results of calling a provided function on every element in the calling array.

e.g. 1) const nums = [1, 2, 3, 4]
Output : [1, 4, 9, 16]

↳ const squares = nums.map(x => x * x)
console.log(squares)

2) const names = ["khushi", "patel"]
Output : ["Khushi", "Patel"]

↳ const capital = names.map((name) => {
    return name[0].toUpperCase() + name.sli
})

function () => {

Miracle
Pg.No :
Date: / /20

3) const pokemon = ["khushi", "Patel"]
Output : [" khushi ", " Patel "]
↳ const para = pokemon.map(mon => `<p>${mon}</p>`)

If one single line in return, we can also write this way

✱ Event Listeners (Handling Events)

→ camelCase

→ With JSX, we pass a function as the event handler, rather than a string

- e.g. In HTML,
  no camelCase
  <button onclick="activate()">
  Activate
  </button>
  a string
- In React
  camelCase
  <button onClick={activate}>
  Activate
  </button>
  function

→ There are many more event listeners in React. See documentation.

Miracle
Pg.No :
Date: / /20

State

⟹ "PROPS" refers to the properties being passed into a component in order to work it correctly, similar to how a function receives parameters (from above)

A component receiving props is not allowed to modify those props i.e. they are immutable.

e.g. function A(a, b) {
return a+b
}
A(1,2)
= 3

function Add(a, b) {
a = 42
return a+b
}
console.log(Add(1,2))
Output : 44

Even if we added a=1, in the func. it is set to 42. that is wrong. We are not allowed to modify it (same in props)

⇒ "STATE" refers to the values that are managed by the component, similar to variables declared inside a function.

Anytime you have changing values that should be saved / displayed, you'll likely be using state.

⇒ Prop vs. State Quiz:

1) How would you describe the concept of "state"?

→ A way for React to remember saved values from within a component. This is similar to declaring variables from within a component, with few added bonuses.

2) When would you want to use props instead of state?

→ Anytime you want to pass data into a component so that component can determine what will get displayed on the screen.

3) When would you want to use state instead of props?

→ Anytime you want a component to maintain some values from within a

component. (And "remember" these values even when React re-renders the component)

4) What does "immutable" mean? Are props immutable? Is state immutable?

→ Unchanging
Props are immutable (like they are mutable if we try to set it to somethin else, but you should not change props)

State is not immutable. (i.e. its changing)

⟹ useState → is a ~~hook~~

import React, {useState} from "react"
→ Agar aai declare Kariye to niche React lakhvani nai jarur.

or React.useState()

↳ e.g

```
const result = React.useState()
consol.log(result)
```
↳ [undefined, f()] → We get an array one undefined and other f() is func

```
const result = React.useState("Hello")
console.log(result)
```
↳ ["Hello", f()]

→ Destructuring ~~useArray~~ useState

```
const [result, func] = React.useState("Hi")
```

result ↑ value of state (undefined)
func ↑ function (f())

```
console.log(result)
```
=> Hi

→ Changing state

The function allows us to make changes so that React can handle the changes accordingly.

→ Example

Create a counter:

```
const [count, setCount] = React.useState(0)

function Increment() {
    setCount((prevCount) => {
        return prevCount + 1
    })
}

function Decrement() {
    setCount(prevCount => prevCount - 1)
}
```

Callback func.

One line function return value hai toh aaru lakhai direction without { } etc.

html:

<button onClick={Increment}> + </button>

<h1> {count} </h1>

<button onClick={Decrement}> - </button>

*/ Note :

Whenever you need the old value of state to determine the new value of state, you should pass a callback function to your state setter function instead of using state directly.

This callback function will receive the old value of state as its parameter, which you can then use to determine your new value of state.

⟹ Changing state quiz :

1) What are the two ways to pass in a state setter func. (e.g. setCount)

→ (1) New value of state (setCount(42))

(2) Callback func.
Whatever callback func. returns ==> new value of state.

2) When to use option (1)?

→ Whenever you don't need the previous value of state to determine what the new value of state should be.

3) When to use option (2)?

→ Whenever you DO need the previous value to determine the new value.

✱ About hooks (Web Dev Simplified)

→ useState is a hook.

→ We cannot use hooks inside a class component. Only inside functions we can use.

→ We cannot use it inside if conditions, for loops (inside any nested condition)

→ Call it in order.

e.g. function App() {
useState() → 1st aa call thase
useState() → 2nd aa (1st pachi j)
}