5_TALK_IS_CHEAP

return ke sath ke sath bracket tabhi need hai jab return ke baad multiple lines code ho

{} // inside this we can write any piece of JS

JSX is not mandatory in react

even ES6 is not mandatory

we can also use react only in footer

<title/> instead of this we can also use <title></title> example is shown below

```
import React from "react";
import ReactDOM from "react-dom/client";


const rootElement = document.getElementById('root');


const Heading=(<h1>this is food app</h1>);


const Header=()=>{

  return (
    <div>
        {Heading} // we cannot use <Heading/> or <Heading></Heading> as Heading is not a component
    </div>
  )
}
const root = ReactDOM.createRoot(rootElement);


root.render(<Header></Header>);
```

lets build the food website

structure look like this

comments inside components

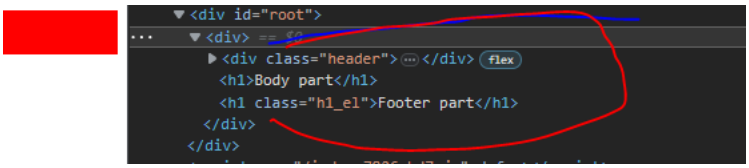
```
const Header_components=()=>{
    return(
        <div>
            <Title/>
            {/* nav bar */}

        </div>
    );
}
```

JSX should have only one parent lets understand with example

```
const jsx=<h1>Thisis h1</h1><h2>this is h2</h2> // invalid syntax
// in order to correct we can write it inside div

const jsx=<div><h1>Thisis h1</h1><h2>this is h2</h2></div>// valid syntax
```



as from above image we are able to inside root there is another div it because JSX has only one parent element inside which we can bundle things like this

```
const App_layout=()=>{
    return (
<div>
    <Header_components></Header_components>
    <Body></Body>
    <Footer></Footer>
</div>
    )
}


const root = ReactDOM.createRoot(rootElement);


root.render(<App_layout/>)
```

but inside root another div aana ye good practice nhi hai

so to solve this issue we have react.fragment

In React, a fragment is a built-in feature that allows you to group multiple elements together without adding an extra DOM element. It's useful when you need to return multiple elements from a component's render method without introducing an additional wrapping element.

like in this example

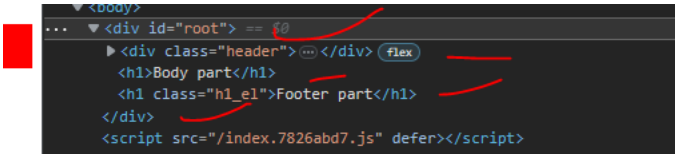
```
const App_layout=()=>{
    return (
<React.Fragment>
    <Header_components></Header_components>
```

```
    <Body></Body>
    <Footer></Footer>
</React.Fragment>
    )
}


const root = ReactDOM.createRoot(rootElement);


root.render(<App_layout/>)
```

nows lets see the sturcture of above code inside browser



so there is no extra div inside root header ka div hai vo bhi isliye kyunnki **<Header_components></Header_components>** div return kr raha hai

so **React.Fragment** is like empty tag

instead of **React.Fragment we can also write<> </>**

```
const App_layout=()=>{
    return (
<>
    <Header_components></Header_components>
    <Body></Body>
    <Footer></Footer>
</>
    )
}
```

so we can use React.Fragment like this **<> </>**

but problem with empty tag is we can write any id , css etc property of this **<> </>**

the `<React.Fragment>` component itself does not accept the `className` attribute. It is not a regular DOM element.

the **style** inside react object mangta hai

this below is the inline css in react

```
// inline styling to body part
const body_css={
    backgroundColor:"blue"
}


// BODY PART
const Body=()=>{
    return (
        <h1 style={body_css}>Body part</h1>
    )
}
```

homework⇒ can we use React.Fragment inside React.Fragment?

yes we can use React.fragment inside React.Fragment

```
const burger_king={
    Name:"Burger King",
    Image:burger,
    Cusines:["Burger","American"],
    Rating:"4.2"
}

// restraunt card which is inside body
const Restraunt_card=()=>{
   return  <div className="card">
        <img src={burger_king.Image} alt="burger"/>
        <h2>{burger_king.Name}</h2>
        <h3>{burger_king.Cusines.join(", ")}</h3>
        <h4>{burger_king.Rating} Stars</h4>
    </div>
}
```

this above coding is like hard coding practice like burger ke liye alag object hai agar pizza ayega toh uske liye alag object bnana padega so it is not good

so we should use **config driven UI**

A config-driven UI (User Interface) refers to a user interface design approach where the behavior and appearance of the interface are primarily determined by a configuration file or set of configuration parameters. Rather than hard-coding all the details of the UI within the application's codebase, the UI elements, their layout, styling, and behavior are defined through a separate configuration file or database.

HOMEWROK⇒ WHAT IS OPTIONAL CHAINING ?.

The optional chaining operator `?.` is used to access properties and methods of an object. If the property or method exists, it is accessed as usual. However, if any intermediate property or method in the chain is `null` or `undefined`, the expression short-circuits and evaluates to `undefined`, instead of throwing a `TypeError`.

<Restraunt_card **restraunt={restaurantList[0]}**></Restraunt_card>

this highlight part in above code is called props

props ⇒ properties

function x(a,b){ // a and b is parameters

}

x(”abcd”,”xyzc”); // abcd and xyzc is called arguments

<Restraunt_card **restraunt={restaurantList[0]} hello=”world”**></Restraunt_card> passing multiple props

Optional chaining is a feature introduced in JavaScript (starting with ECMAScript 2020) that allows you to safely access nested properties or methods of an object without causing an error if any intermediate property is null or undefined. It helps to simplify the code and avoid unnecessary null or undefined checks.

```
const user = {
  name: 'John',
  address: {
    city: 'New York',
    zipcode: 12345
  }
};

// Accessing nested properties without optional chaining
const city = user.address.city; // 'New York'
const street = user.address.street; // TypeError: Cannot read property 'street' of undefined

// Accessing nested properties with optional chaining
const cityWithChaining = user?.address?.city; // 'New York'
const streetWithChaining = user?.address?.street; // undefined
```

Object destructuring is a feature in JavaScript that allows you to extract individual properties from an object and assign them to variables in a concise and structured way. It provides a more convenient syntax for extracting and working with specific values from an object, without the need to access the object's properties using dot notation.

```
const user = {
  name: 'John',
  age: 30,
  address: {
    city: 'New York',
    zipcode: 12345
  }
};

// Extracting properties using object destructuring
const { name, age } = user;
console.log(name); // 'John'
console.log(age); // 30

// Extracting nested properties using object destructuring
const { address: { city } } = user;
console.log(city); // 'New York'
```

```
// restraunt card which is inside body
const Restraunt_card=(props)=>{
  console.log("props is ",props);
   return  <div className="card">
         <img
        src={
          "https://res.cloudinary.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_508,h_320,c_fill/" +
         props.restraunt.data.cloudinaryImageId
        }
        ></img>
        <h2>{props.restraunt.data.name}</h2>
        <h3>{props.restraunt.data.cuisines.join(", ")}</h3>
        <h4>{props.restraunt.data.lastMileTravelString} minutes</h4>
    </div>
}


// BODY PART
const Body=()=>{
    return (
       <div className="rest_card_body">
        <Restraunt_card restraunt={restaurantList[0]}></Restraunt_card> {/*we can say passing restrunnt argument   and this line is alterna
        <Restraunt_card restraunt={restaurantList[1]}></Restraunt_card>
        <Restraunt_card restraunt={restaurantList[2]}></Restraunt_card>
```

```
        <Restraunt_card restraunt={restaurantList[3]}></Restraunt_card>
        <Restraunt_card restraunt={restaurantList[4]}></Restraunt_card>
        <Restraunt_card restraunt={restaurantList[5]}></Restraunt_card>

       </div>
    )
}
```

now we are going to destruct object

```
// restraunt card which is inside body
const Restraunt_card=({restraunt})=>{
  const {name,cuisines,lastMileTravelString,cloudinaryImageId}=restraunt.data;
   return  <div className="card">
         <img
        src={
          "https://res.cloudinary.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_508,h_320,c_fill/" +
        cloudinaryImageId
        }
        ></img>
        <h2>{name}</h2>
        <h3>{cuisines.join(", ")}</h3>
        <h4>{lastMileTravelString} minutes</h4>
    </div>
}


// BODY PART
const Body=()=>{
    return (
       <div className="rest_card_body">
        <Restraunt_card restraunt={restaurantList[0]}></Restraunt_card> {/*we can say passing restrunnt argument   and this line is alterna
        <Restraunt_card restraunt={restaurantList[1]}></Restraunt_card>
        <Restraunt_card restraunt={restaurantList[2]}></Restraunt_card>
        <Restraunt_card restraunt={restaurantList[3]}></Restraunt_card>
        <Restraunt_card restraunt={restaurantList[4]}></Restraunt_card>
        <Restraunt_card restraunt={restaurantList[5]}></Restraunt_card>

       </div>
    )
}
```

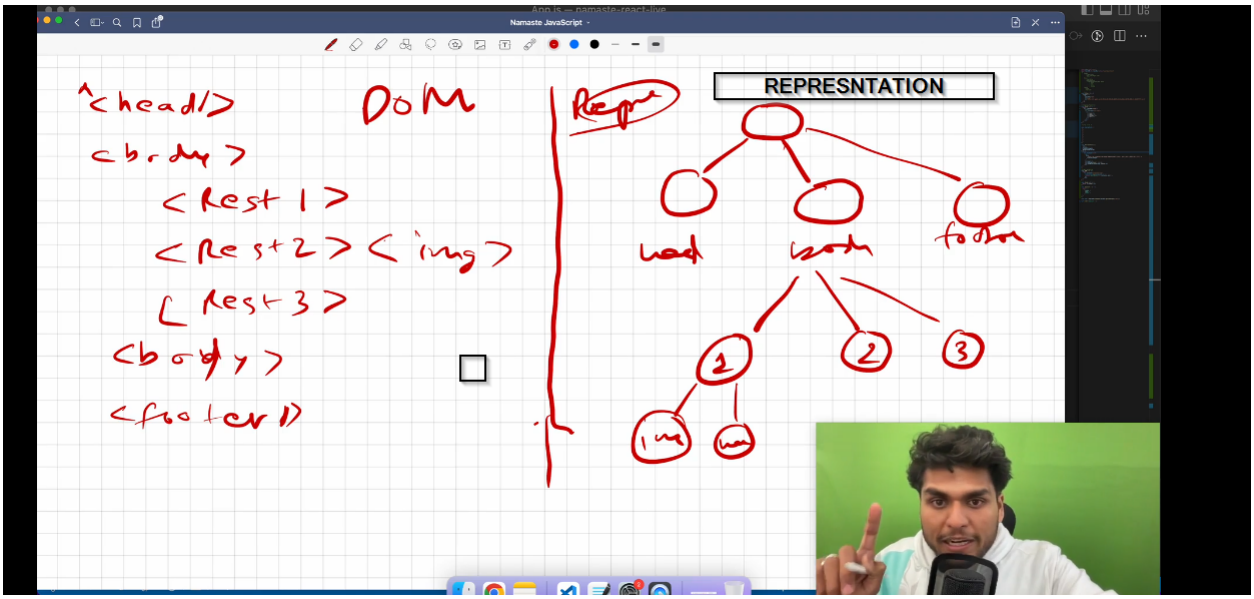
in industry we dont use for loop we use map function

homemwork ⇒ diff between for each and for loop

```
const array=[1,2,3,4];
array.forEach(function(element, index) {
    console.log('Element:', element, 'Index:', index);
  });

/*
In general, forEach is often preferred when you simply want to iterate over each element of an array and perform some action or side effect
 It provides a more expressive and concise syntax.
However, if you need more control over the iteration process, need to break the loop early, or want to accumulate values or create a new ar
*/
```

we keep represntation of DOM with us is called virtual DOM



why we need virtual DOM?

due to reconsillation **READ MORE**

In React, reconciliation refers to the process of comparing and updating the virtual DOM (VDOM) to reflect changes in the component tree and efficiently updating the actual DOM.

When you render a React component, it creates a virtual representation of the component's UI structure known as the virtual DOM. This virtual DOM is a lightweight copy of the actual DOM and is used by React to determine the minimal set of changes required to update the real DOM.

During reconciliation, React performs a diffing algorithm to identify the differences between the previous virtual DOM and the new virtual DOM. It analyzes the component tree, compares the old and new elements, and determines the most efficient way to update the actual DOM. This diffing process helps minimize unnecessary DOM manipulations, resulting in improved performance.

react use an reconsillation to diff from one DOM tree to another VIRTUAL tree and it determeines what need to change and what not to do

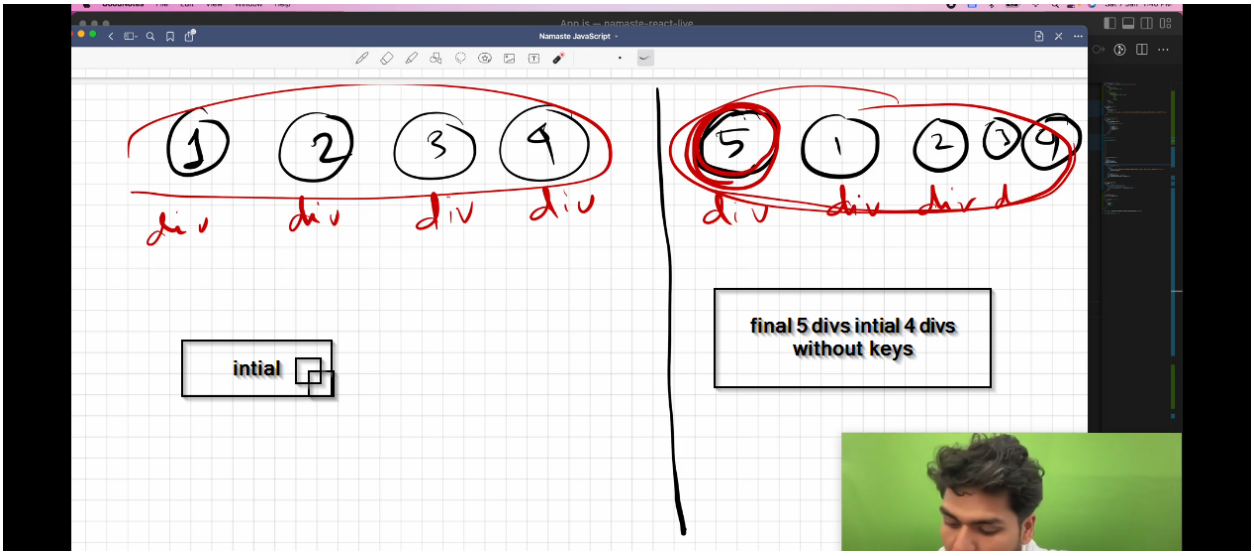
for example if we build project contain ⇒ header, body,footer

now we add some text to footer so reconsillation algorithm to check the diff between virtual dom and dom and see header and body is same but a small change in footer so it updates footer only

why we need keys in react?

now in below example intialy have 4 divs and we add 1 div (5) in intial now react knows there are 5 divs but reacct does not know which is new intrdouced div so react rerender everything

so to solve this issue we use key so react will able to indnentity new div and put it on the front without rerender everything



why react is fast?

due to virtual DOM and reconsollation which has diff algorithm it knows the only portion that need to update

what is react fibre? ⇒ HOMEWORK

React Fiber (also known as React reconciler or React Fiber reconciler) is a complete rewrite of the React reconciliation algorithm and the core algorithmic engine of React. It was introduced in React 16 as an internal reimplementation of React's reconciliation process.

The main goal of React Fiber is to provide a more efficient and incremental rendering process, enabling better performance, responsiveness, and support for more advanced features like concurrent rendering and error boundaries.

HOMEWORK

is index is a valid key? ⇒ true

but why we don't use it?

index array indexing vala hi hai

but if we dont have any key than we can use index as key

prefenece order for key

nokey< index key< unique id

why we dont prefer index as key in js?

The index of an item in a list can change if elements are added, removed, or reordered. If the index changes, React may incorrectly identify different items as the same based on the key. This can cause unexpected behavior, such as components losing their state or not updating correctly.

if childrens are div and another image than we dont need key

keys neeed when children are of same type

what are props in react?

In React, "props" (short for "properties") are a way to pass data from a parent component to a child component. Props are read-only and are used to configure and customize child components.

ways to use props

```
// ParentComponent.js
import React from 'react';
import ChildComponent from './ChildComponent';

function ParentComponent() {
  const greeting = 'Hello';

  return <ChildComponent message={greeting} />; // sending props to childcomponent
}

// ChildComponent.js
import React from 'react';

function ChildComponent(props) { // receiving props
  return <div>{props.message}</div>;
}
```

passing props through spreadoperator

```
function ParentComponent() {
  const props = { message: 'Hello' };

  return <ChildComponent {...props} />;
}
```

Default Props: You can define default values for props using the `defaultProps` static property on a component. These default values are used when a prop is not explicitly provided.

```
function ChildComponent(props) {
  return <div>{props.message}</div>;
}

ChildComponent.defaultProps = {
  message: 'Default Message',
};
```

Passing Functions as Props: In addition to data, you can also pass functions as props to enable communication and interaction between components.

```
function ParentComponent() {
  const handleButtonClick = () => {
```

```
    console.log('Button clicked');
  };

  return <ChildComponent onClick={handleButtonClick} />;
}

function ChildComponent(props) {
  return <button onClick={props.onClick}>Click me</button>;
}
```