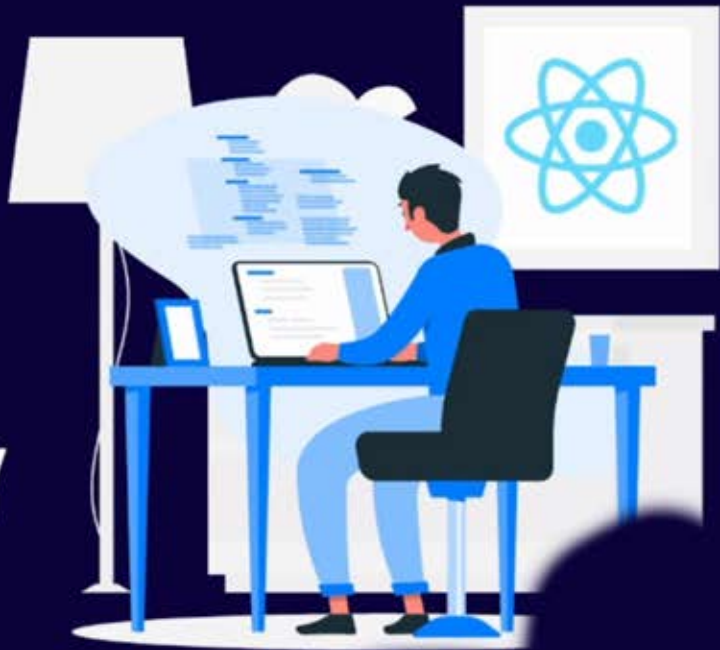




# REACT JSX, PROPS, STATE, HOOKS AND ATTRIBUTE VS PROPERTY



## What is JSX?

- JavaScript XML - JSX
- JSX is a syntax extension for React JS
- Easier to read and write
- Gets transpiled to JavaScript with **Babel**

## Rules of JSX

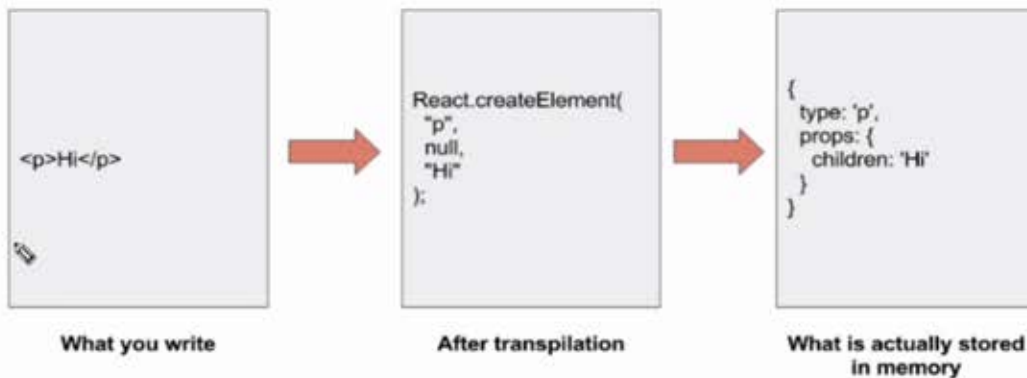
- JSX looks like HTML, but under the hood it is transformed into plain JavaScript objects. This explains why you also can't return two JSX tags without wrapping them into another tag `<div></div>` or a Fragment `<></>`.
- JSX requires tags to be explicitly closed: self-closing tags like `<img>` must become `<img />`, and wrapping tags must be written as `<li>oranges</li>`.
- "class" attribute becomes "className". Properties are written in camelCase
- onclick → onClick, tabindex → tabIndex

## **Why can't browsers read JSX?**

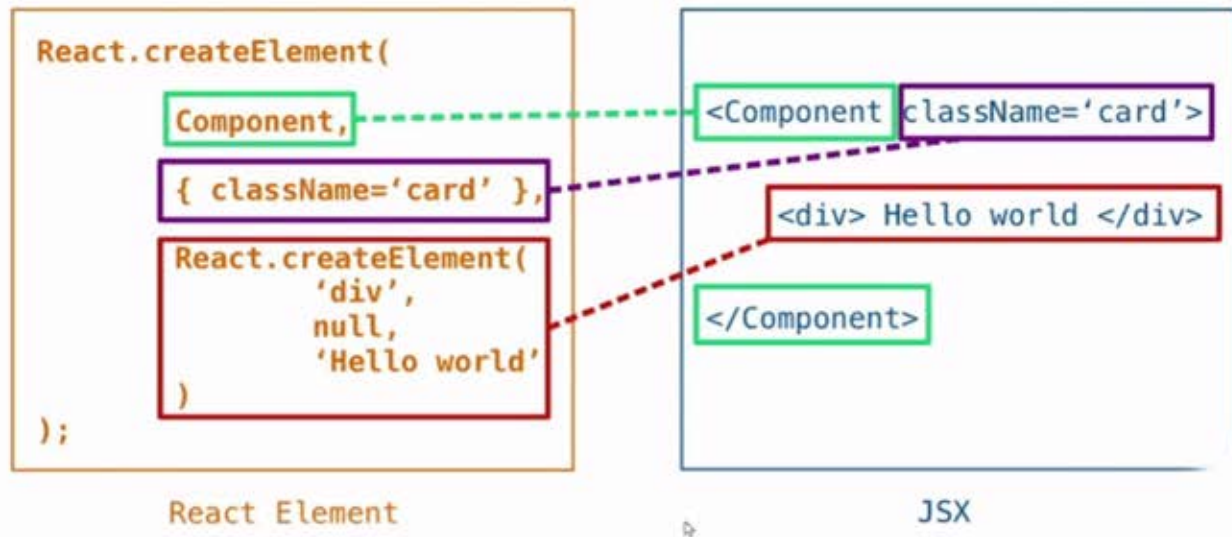
- Browsers can only read JavaScript objects.
- JSX is not a regular JavaScript object
- To enable a browser to read JSX, we need to transform JSX file into a JavaScript object using JSX transpiler like Babel and then pass it to the browser.

# How JSX is converted?

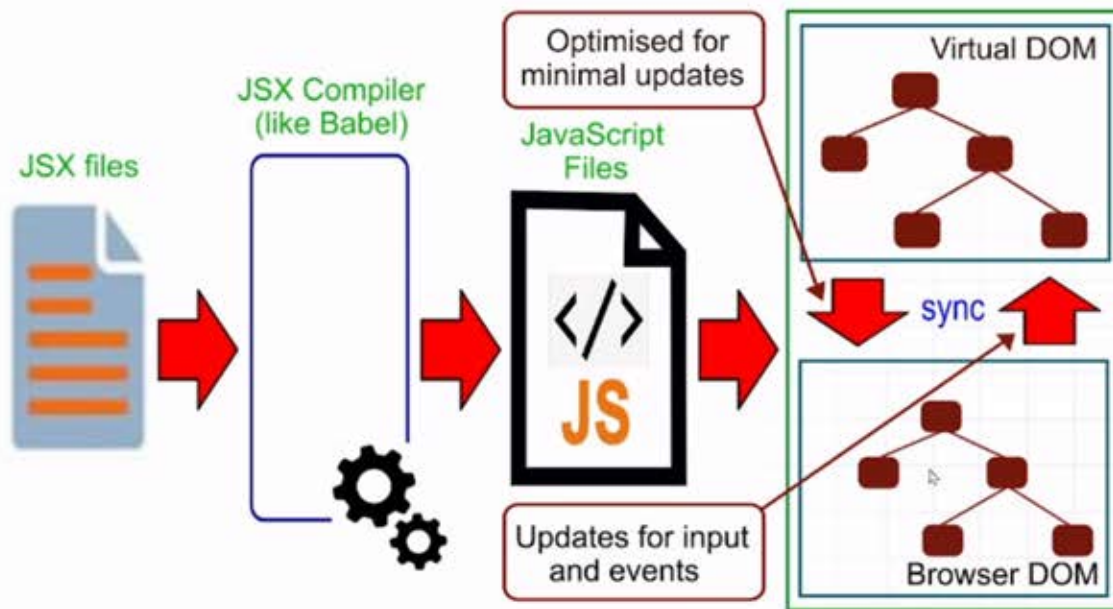
## JSX



# React Element Vs JSX

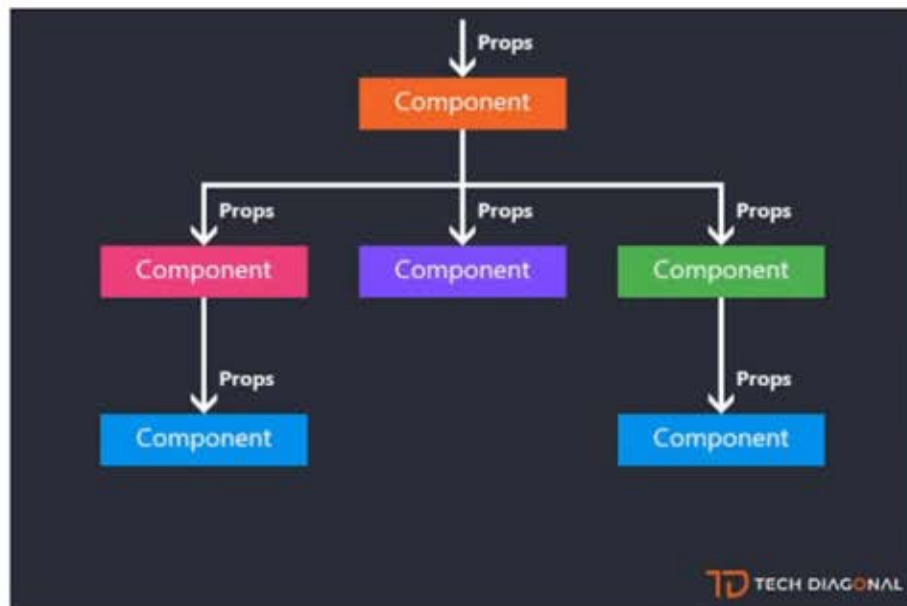


# What happens behind the scene?



# What are props?

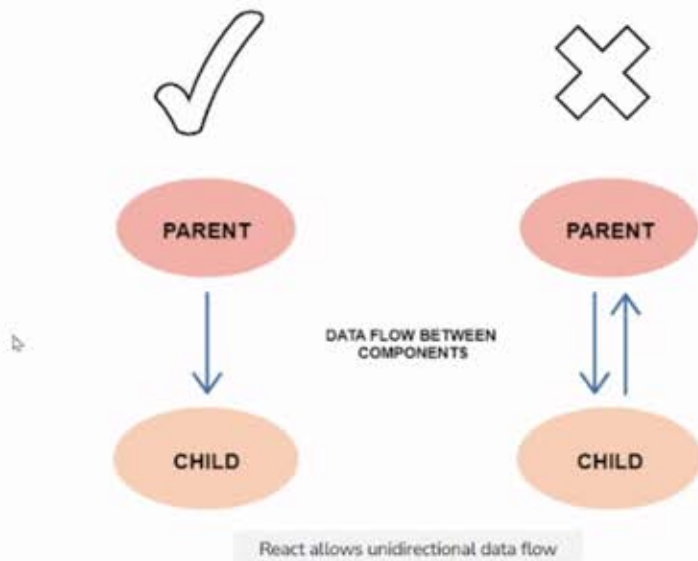
We use props in React to pass data from one component to another (from a parent component to a child component(s)).



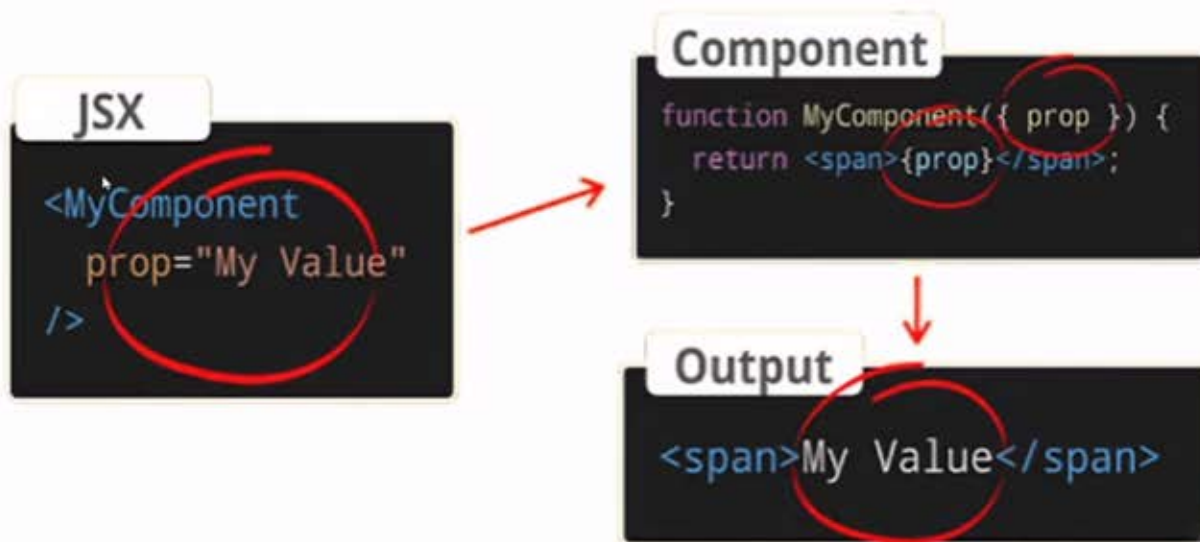


# Props

Props are unidirectional.



# Props



# Values of Props

1. String literals:

```
<MyComponent prop="My String Value">
```

2. Template literals with variables:

```
<MyComponent prop={`My String Value ${myVariable}`}>
```

3. Number literals:

```
<MyComponent prop={42} />
```

4. Boolean literals:

```
<MyComponent prop={false} />
```

# Values of Props

5. Plain object literals:

```
<MyComponent prop={{ property: 'Value' }} />
```

6. Array literals:

```
<MyComponent prop={['Item 1', 'Item 2']} />
```

7. JSX:

```
<MyComponent prop=<Message who="Joker" /> />
```

8. Variables having any kind of value:

```
<MyComponent prop={myVariable} />
```

## Multiple Props

We can use as many props as we like.

```
function Message({ greet, who }) {  
  return <div>{greet}, {who}!</div>;  
}
```

```
// Render  
<Message greet="Welcome" who="Aliens" />  
  
// Output  
<div>Welcome, Aliens!</div>
```

## Default Props

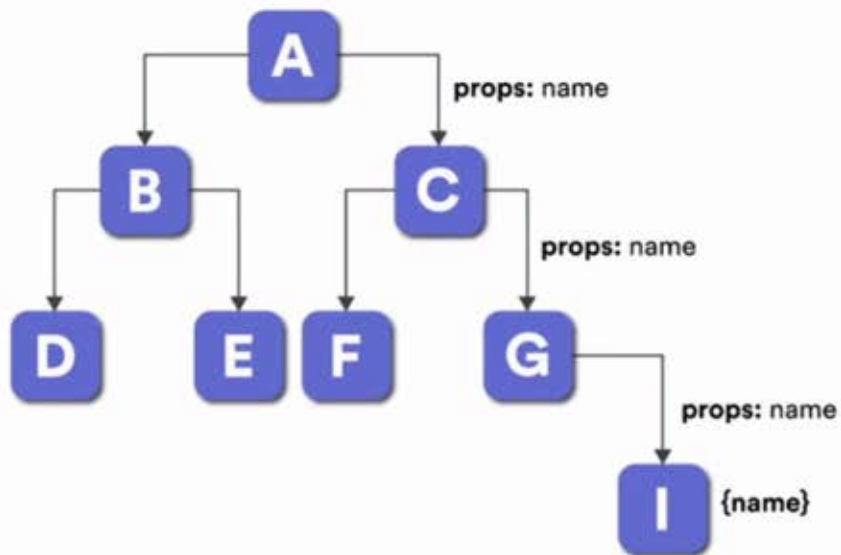
```
function HelloOptional({ who = 'Unknown' }) {  
  return <div>Hello, {who}</div>;  
}
```

```
// Render  
<HelloOptional />
```

```
// Output  
<div>Hello, Unknown!</div>
```

## Prop Drilling

Prop drilling is a method where we pass a props with another component with the help of all the components that come between them.



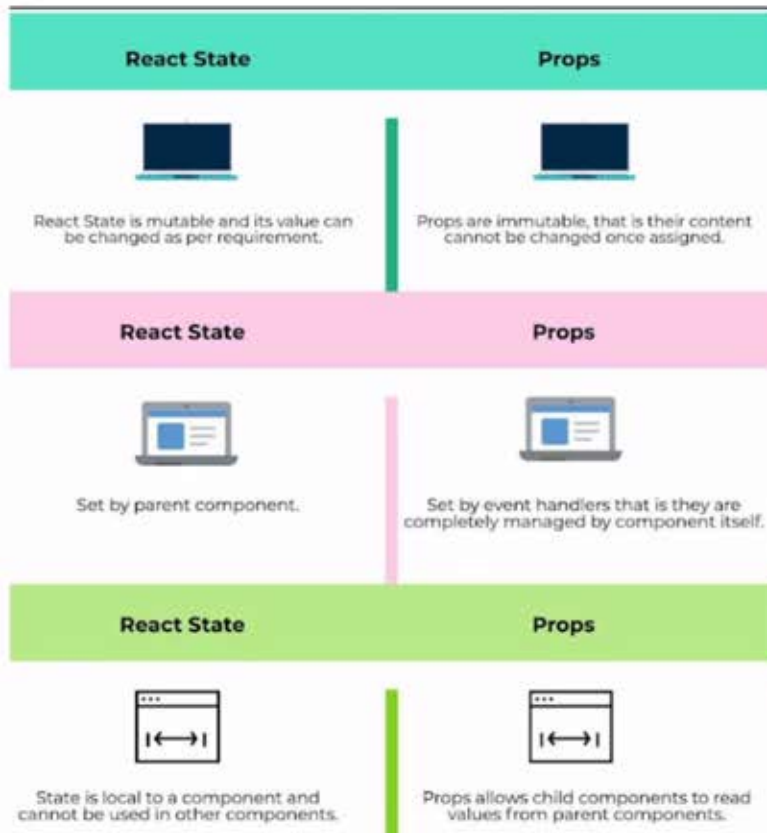
# What is a state?

- The state is a built-in React object that is used to contain data or information about the component.
- A component's state can change over time; whenever it changes, the component re-renders.
- The change in state can happen as a response to user action or system-generated events





# State vs Props



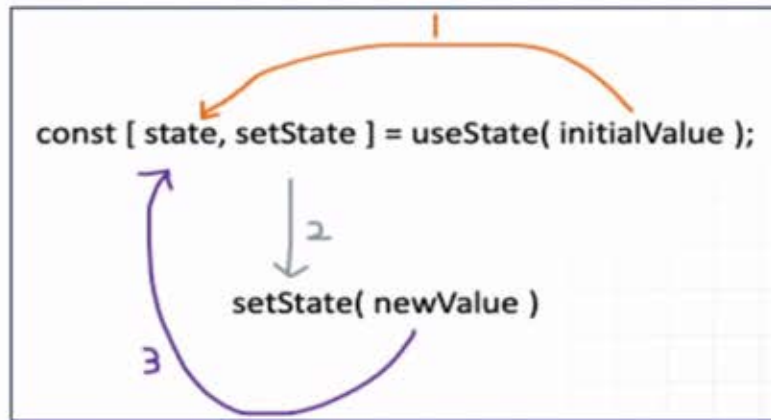
# Hooks

- Hooks were added to React in version 16.8.
- Hooks allow function components to have access to state and other React features.
- Hooks allow us to "hook" into React features such as state and lifecycle methods.

## **useState Hook**

- The useState Hook allows us to track state in a function component.
- To use the useState hook, you need to know a few things.
  1. You must import it from the React library.
  2. You must invoke it inside a React component

## useState Syntax



**const [state, setState] = useState(initialValue )**



The name of  
your state



The function you'll  
eventually use to  
change the value of this



The initial value  
of your state

## useState Code Example

```
import { useState } from 'react'

function App() {
  const [state, setState] = useState(1)

  return (
    <section>
      <div>{state}</div>
      <button onClick={() => setState(state + 1)}>More</button>
      <button onClick={() => setState(state - 1)}>Less</button>
    </section>
  )
}
```

## useEffect Hook

- The `useEffect` Hook allows you to perform side effects in your components. Some examples of side effects are: fetching data, directly updating the DOM, and timers.
- To use the `useEffect` hook, we need to know a few things.
  1. We import `useEffect` from "react"
  2. We call it above the returned JSX in our component
  3. We usually pass two arguments: a function and an array

## useEffect Syntax

```
useEffect( () => {
```

```
  // perform a side effect
```

```
}, [ ] );
```


### **effect function:**

we only want to run this function once, when the component mounts

### **dependency list:**

an empty list causes the effect to run once, when the component first mounts

## useEffect Code Example



```
import { useEffect } from 'react';

function User({ name }) {
  useEffect(() => {

    document.title = name;

  }, [name]);

  return <h1>{name}</h1>;
}
```



# Interview Questions

PAGE 26

1. What is React?
2. What is the difference between virtual dom and real dom?
3. What is JSX?
4. What is the difference between state and props?



## Interview Questions

PAGE 27

7. What are react lifecycle methods?
8. What are hooks? Tell us the role of useEffect and useState.
9. What is diff algorithm?
10. What is the difference between attribute and property?



# Thank you!