Doubts:

***REACT JS:***

1)Pros: This means properties, which is used to contain the objects component in the JSX attributes.

Ex. function yo(props){ //content }

ReactDOM.render( <Header roll:”10” name: “deba” )

, here <Header /> is an user defined component and also called JSX attributes & “this points to the header only, and root & name are the attributes, which together considered as a single object and passes to the “props” object in function.

Above pros contains thee attributes as ,

Props = {roll: “10”,name: “deba” };

1)What is the difference between Real DOM and virtual DOM?

# Ans. React: The Virtual DOM

Fighting Wasteful DOM Manipulation

**The Problem**

[DOM manipulation](https://www.codecademy.com/courses/build-interactive-websites/lessons/javascript-dom/exercises/document) is the heart of the modern, interactive web. Unfortunately, it is also a lot slower than most JavaScript operations.

This slowness is made worse by the fact that **most JavaScript frameworks update the DOM much more than they have to.**

As an example, let’s say that you have a list that contains ten items. You check off the first item. Most JavaScript frameworks would rebuild *the entire list*. That’s ten times more work than necessary! Only one item changed, but the remaining nine get rebuilt exactly how they were before.

Rebuilding a list is no big deal to a web browser, but modern websites can use huge amounts of DOM manipulation. Inefficient updating has become a serious problem.

To address this problem, the people at React popularized something called the *virtual DOM.*

**The Virtual DOM**

In React, for every [DOM object](http://eloquentjavascript.net/13_dom.html), there is a corresponding “virtual DOM object.” A virtual DOM object is a *representation* of a DOM object, like a lightweight copy.

A virtual DOM object has the same properties as a real DOM object, but it lacks the real thing’s power to directly change what’s on the screen.

Manipulating the DOM is slow. Manipulating the virtual DOM is much faster, because nothing gets drawn onscreen. Think of manipulating the virtual DOM as editing a blueprint, as opposed to moving rooms in an actual house.

**How it helps**

When you render a JSX element, every single virtual DOM object gets updated.

This sounds incredibly inefficient, but the cost is insignificant because the virtual DOM can update so quickly.

Once the virtual DOM has updated, then React compares the virtual DOM with a virtual DOM *snapshot* that was taken right before the update.

By comparing the new virtual DOM with a pre-update version, React figures out *exactly which virtual DOM objects have changed.* This process is called “diffing.”

Once React knows which virtual DOM objects have changed, then React updates those objects, *and only those objects,* on the real DOM. In our example from earlier, React would be smart enough to rebuild your one checked-off list-item, and leave the rest of your list alone.

This makes a big difference! React can update only the necessary parts of the DOM. React’s reputation for performance comes largely from this innovation.

In summary, here’s what happens when you try to update the DOM in React:

1. The entire virtual DOM gets updated.
2. The virtual DOM gets compared to what it looked like before you updated it. React figures out which objects have changed.
3. The changed objects, and the changed objects only, get updated on the *real* DOM.
4. Changes on the real DOM cause the screen to change.

If you’d like to learn more about the virtual DOM, [here’s a good place to start](http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/).

Link:- https://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/

### 2. What is JSX?

JSX stands for JavaScript XML.  
It allows us to write HTML inside JavaScript and place them in the DOM without using functions like appendChild( ) or createElement( ).  
As stated in the official docs of React, JSX provides syntactic sugar for React.createElement( ) function.  
**\*\*Note- We can create react applications without using JSX as well.**  
Let’s understand how JSX works:  
  
**Without using JSX,** we would have to create an element by the following process:

**const** text = React.createElement('p', {}, 'This is a text');

**const** container = React.createElement('div','{}',text );

ReactDOM.render(container,rootElement);

**Using JSX,** the above code can be simplified:

const container = (

<div>

<p>This is a text</p>

</div>

);

ReactDOM.render(container,rootElement);

As one can see in the code above, we are directly using HTML inside JavaScript.

### 3. What are the differences between functional and class components?

Before the introduction of Hooks in React, functional components were called stateless components and were behind class components on feature basis. After the introduction of Hooks, functional components are equivalent to class components.  
Although functional components are the new trend, the react team insists on keeping class components in React. Therefore, it is important to know how these both components differ.  
On the following basis let’s compare functional and class components:

* Decalaration  
  Functional components are nothing but JavaScript functions and therefore can be declared using an **arrow function** or the **function** keyword:
* function card(props){
* return(
* <div className="main-container">
* <h2>Title of the card</h2>
* </div>
* )
* }
* const card = (props) =>{
* return(
* <div className="main-container">
* <h2>Title of the card</h2>
* </div>
* )
* }

Class components on the other hand, are declared using the ES6 class:

class Card extends React.Component{

constructor(props){

super(props);

}

render(){

return(

<div className="main-container">

<h2>Title of the card</h2>

</div>

)

}

}

* Handling props  
  Let’s render the following component with props and analyse how functional and class components handle props:
* <StudentInfo name="Vivek" rollNumber="23" />

In functional components, the handling of props is pretty straight forward. Any prop provided as an argument to a functional component, can be directly used inside HTML elements:

function StudentInfo(props){

return(

<div className="main">

<h2>{props.name}</h2>

<h4>{props.rollNumber}</h4>

</div>

)

}

In the case of class components, props are handled in a different way:

**class** StudentInfo **extends** React.Component{

constructor(props){

**super**(props);

}

render(){

**return**(

<div className="main">

<h2>{**this**.props.name}</h2>

<h4>{**this**.props.rollNumber}</h4>

</div>

)

}

}

As we can see in the code above, **this** keyword is used in the case of class components.

* Handling state  
  Functional components use React hooks to handle state.  
  It uses the **useState** hook to set state of a variable inside the component:
* function ClassRoom(props){
* let [studentsCount,setStudentsCount] = useState(**0**);
* const addStudent = () => {
* setStudentsCount(++studentsCount);
* }
* **return**(
* <div>
* <p>Number **of** students **in** **class** room: {studentsCount}</p>
* <button onClick={addStudent}>Add Student</button>
* </div>
* )
* }

Since useState hook returns an array of two items, the first item contains the current state, and the second item is a function used to update the state.  
In the code above, using array destructuring we have set the variable name to studentsCount with a current value of “0” and setStudentsCount is the function that is used to update the state.  
For reading the state, we can see from the code above, the variable name can be directly used to read the current state of the variable.  
We cannot use React Hooks inside class components, therefore state handling is done very differently in a class component:  
Let’s take the same above example and convert it into a class component:

**class** ClassRoom **extends** React.Component{

constructor(props){

**super**(props);

**this**.state = {studentsCount : **0**};

**this**.addStudent = **this**.addStudent.bind(**this**);

}

addStudent(){

**this**.setState(**(prevState)=>**{

**return** {studentsCount: prevState.studentsCount++}

});

}

render(){

**return**(

<div>

<p>Number **of** students **in** **class** room: {**this**.state.studentsCount}</p>

<button onClick={**this**.addStudent}>Add Student</button>

</div>

)

}

}

In the code above, we see we are using **this.state** to add the variable studentsCount and setting the value to “0”.  
For reading the state, we are using **this.state.studentsCount**.  
For updating the state, we need to first bind the addStudent function to **this**. Only then, we will be able to use the **setState** function which is used to update the state.

### 4. What is the virtual DOM? How does react use the virtual DOM to render the UI?

As stated by the react team, virtual DOM is a concept where a virtual representation of the real DOM is kept inside the memory and is synced with the real DOM by a library such as ReactDOM. The main thing is inside the index js and the other functional or class components created and then exported to index.js part by part, which are later imported inside index.js, this all happens in virtual DOM, where the changes are done, and later these overall changes are reflected in the real DOM.

The changes done in the virtual DOM is a lot faster then Real Dom, because changes are done in small small components and are then reflected in real DOM.

Before React, Developers directly manipulated the DOM elements which resulted in frequent DOM manipulation, and each time an update was made the browser had to recalculate and repaint the whole DOM view each time according to the particular CSS of the page, which made the total process to consume a lot of time. As a betterment, React brought **Virtual DOM** which is a copy of the actual DOM that is used to hold the updates made by the user and finally reflect it over to the original Browser DOM at once consuming much lesser time.

If we don’t use react then when the changes are made in real DOM, then whole DOM is reloads the sections or whole DOM everytime.

Use of ReactDOM?

ReactDOM is a package that provides DOM specific methods that can be used at the top level of a web app to enable an efficient way of managing DOM elements of the web page. ReactDOM provides the developers with an API containing the following methods and a few more.

* render()
* findDOMNode()
* unmountComponentAtNode()
* hydrate()
* createPortal()