Chapter 04 Assignment Theory

1. **Is JSX mandatory for React?**
   No. JSX is not mandatory. React can be included via CDN and used in plain JS. For example: in JS we can use React.createElement()

2. **Is ES6 mandatory for React?**
   No. Without ES6 react can still be used. For example:

```
var Counter = createReactClass({
  getInitialState: function() {
    return {count: this.props.initialCount};
  },
  // ...
});
```

3. **How can I write comments in JSX?**
   In React, we can write JS within { }, so comments can be written as:
   { // const abc = "hello" } – for single line comment use, **//**
   { / const abc = "hello" */ }* – for multi-line comment use, **/* */**

4. **What is <React.Fragment></React.Fragment> and <></> ?**
   In React, whenever you want to render something on the screen, you need to use a **render** method inside the component. This render method can return **single** elements or **multiple** elements. The render method will only render a single root node inside it at a time. To return more than one element, and you don't want the DOM to have extra div, we can use **<React.Fragment></React.Fragment>** . The short form of same is **<> </>.**

5. **What is Virtual DOM?**
   It is a virtual representation of actual DOM, like a lightweight copy. The browser constructs a logical tree-like structure from the HTML for the user to see the requested page in the client.

6. **What is Reconciliation in React?**
   The mechanism of diffing one tree with another to determine which parts are required to be changed and then update the original DOM with it is called Reconciliation in react.

7. **What is React Fiber?**
   React Fiber is the new reconciliation algorithm in React 16. This new reconciliation algorithm from React is called Fiber Reconciler.
   The main goals of the Fiber reconciler are incremental rendering, better or smoother rendering of UI animations and gestures, and responsiveness of the user interactions. The reconciler also allows you to divide the work into multiple chunks and divide the rendering work over multiple frames. It also adds the ability to define the priority for each unit of work and pause, reuse, and abort the work.

8. **Why we need keys in React? When do we need keys in React?**
   keys serve as identification for an element just like how passports are used to identify people.
   We need keys because of React's Diffing Algorithm. The diffing algorithm compares the new virtual DOM with the old DOM at each level of the component tree, starting from the root node.
   For lists, React will recurse on both their children simultaneously, find any differences, then patch them to the real DOM if there are any.
   But, what if we add a new element at the beginning? This will result in React re-rendering every single <li> to the real DOM because it doesn't realize that it can simply add abc to the beginning of the list.
   This inefficiency can cause problems, especially in larger apps. Hence, keys provide a simple solution to this issue.

9. **Can we use index as keys in React?**
   Index can be used as key only when:
   a. Data is static
   b. You know reordering will not happen
   c. You don't have any other option

   Using index might result in performance issues and data binding issues in case reordering in the form of sorting, filtering might happen

10. **What is props in React?**
    In React, the term props stands for properties, which refers to the properties of an object.
    React props are read-only, that is to say, once the data has been passed to its component, it cannot be changed.
    React props can be thought of as rain falling on layers of soil, typically the flow is down, not up. Because of this, the element that uses the props value is generally the child of the component that contains it. In React props are used primarily to display data that does not change, to render data that needs regular updating you'll need state.

11. **What is a Config Driven UI ?**
    A config driven development involves maintaining a config in say JSON format, which can be used to do all the mundane and repetitive tasks of rendering.
    Maintaining a config has a lot of benefits:
    a. First, it provides a generic interface to develop things which help your project scale well.
    b. It saves a lot of development time and effort. Instead of developing new pages or modules in a imperative way by writing the code for each module yourself, you can just configure the module or page.
    c. The config can be decoupled from the frontend code base and hence any modification required in the future won't require a deployment on UI.
    d. Since the config is maintained in the backend as JSON, experimentation by product teams can be done easily by just changing the configuration but yeah

incorporation of a new element or widget would require writing code for it on the UI of course.

e. Centralised code for taking and rendering the config would be more robust since any errors would be centralised to that code only. Contrary to a scenario where we had different codebases for different modules.