**Questions RE: 12 Main Concepts of React:**

**What is JSX?**

* JSX is a syntax extension to JavaScript. Use it with React to describe what the UI should look like.

**Why do you think JSX was created?**

* JSX was created because it’s helpful as a visual aid when working with UI inside the JavaScript code. It also allows React to show more useful error and warning messages.

**What are the benefits and drawbacks of JSX?**

**Benefits:**

* + Makes components/blocks code readable
  + Testable
  + Prevents injection attacks

**Drawbacks:**

* Some people online found it difficult to learn

**What is rendering? How does React update only what’s needed?**

* Unlike browser DOM elements, React elements are plain objects, and are cheap to create. React DOM takes care of updating the DOM to match the React elements.
* React DOM compares the element and its children to the previous one, and only applies the DOM updates necessary to bring the DOM to the desired state.

**How and why do you compose and extract components?**

* Composing a component is when one component refers to another in its output. You can extract components from complex, nested components, and then use composition to refer to those individual parts. This allows for abstraction (displaying essential information and hiding the details) which makes the code easier to read and understand.

**What is a pure function?**

* A function is pure when it does not attempt to change its input, and always returns the same result for the same inputs.
* All React components must act like pure functions with respect to their props.

**What are the rules of working with the state? Why are they such?**

* You must call setState() to change the state of something – you can’t access it directly by saying something like this.state.example = ‘New’; -> the only place you can assign this.state is within the constructor.
* React often bundles multiple setState() calls into a single update for performance reasons.
* Because this.props and this.state may be updated asynchronously, you should not rely on their values for calculating the next state.
* Neither parent nor child components can know if a certain component is stateful or stateless, and they shouldn’t care whether it is defined as a function or a class. This is why state is often called local or encapsulated. It is not accessible to any component other than the one that owns and sets it.

**What are some of the common life cycle methods in React? When would you use them?**

* componentDidMount() and componentWillUnmount()
* The componentDidMount() method runs after the component output has been rendered to the DOM.
* The componentWillUnmount() method clears the component whenever the DOM produced by the component is removed.

**What is conditional rendering? How can you prevent a component from rendering?**

* Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like if or the conditional operator to create elements representing the current state, and let React update the UI to match them.

**How can you render an array of components in React?**

* You can build collections of elements and include them in JSX using curly braces {}.  
  It seems using map() is a good method for extracting the elements from the array to then be used in individual React elements (like list items).

**How can you extract components with keys?**

* Keys only make sense in the context of the surrounding array. For example, if you extract a ListItem component, you should keep the key on the <ListItem /> elements in the array rather than on the <li> element in the ListItem itself.
* A good rule of thumb is that elements inside the map() call need keys.

function NumberList(props) {

const numbers = props.numbers;

const listItems = numbers.map((number) =>

// Correct! Key should be specified inside the array.

<ListItem key={number.toString()}

value={number} />

);

**What are special considerations for forms in React? What are controlled and uncontrolled components? What are the benefits and drawbacks of either?**

* HTML form elements work a little bit differently from other DOM elements in React, because form elements naturally keep some internal state.
* In most cases, it’s convenient to have a JavaScript function that handles the submission of the form and has access to the data that the user entered into the form. The standard way to achieve this is with a technique called “controlled components”.
* It can sometimes be tedious to use controlled components, because you need to write an event handler for every way your data can change and pipe all of the input state through a React component. This can become particularly annoying when you are converting a pre-existing codebase to React, or integrating a React application with a non-React library.

**What is React recommendation on composition, inheritance and specialization?**

* React recommends using composition instead of inheritance to reuse code between components.
* React recommendation: “At Facebook, we use React in thousands of components, and we haven’t found any use cases where we would recommend creating component inheritance hierarchies. Props and composition give you all the flexibility you need to customize a component’s look and behavior in an explicit and safe way. Remember that components may accept arbitrary props, including primitive values, React elements, or functions. If you want to reuse non-UI functionality between components, we suggest extracting it into a separate JavaScript module. The components may import it and use that function, object, or a class, without extending it.”

**Discuss minimal representation of state, where the state is suggested to live and inverse data flow.**

* Use DRY and the following questions to find the minimal representation of state for your app:

1. Is it passed in from a parent via props? If so, it likely isn’t state.
2. Does it remain unchanged over time? If so, it probably isn’t state.
3. Can you compute it based on any other state or props in your component? If so, it isn’t state.

* As for where the state should live, do the following for each piece of state in your app:

1. Identify every component that renders something based on that state.
2. Find a common owner component (a single component above all the components that need the state in the hierarchy).
3. Either the common owner or another component higher up in the hierarchy should own the state.
4. If you can’t find a component where it makes sense to own the state, create a new component solely for holding the state and add it somewhere in the hierarchy above the common owner component.

* Use callback functions to communicate with parent components to update their state.