React Guidelines

1. To declare a variable inside JSX use {}. Any valid JavaScript expression can be put inside JSX.
2. If we need the JSX to be spread over multiple lines to improve readability

```
const element = (
    <h1>
        Hello, {formatName(user)}!
    </h1>
);
```

This will prevent the pitfalls of automatic semicolon insertion

3. Don't put quotes around curly braces when embedding a JavaScript expression in an attribute. You should either use quotes (for string values) or curly braces (for expressions), but not both in the same attribute.
4. React DOM uses camelCase property naming convention instead of HTML attribute names.
5. JSX tags may contain children. Like a div can contain h1, p, div etc inside it.
6. To render a React element into a root DOM node, pass both to ReactDOM.render():
7. React elements are immutable.
8. 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.

So what we mean by this is that if we have any change in our DOM, then React will update the DOM of only the parts that require to be updated to get the DOM into the new state.

9. When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object "props".

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

 , <p> these are all DOM tags. <Welcome > is a custom user-defined component.

10. Always start component names with a capital letter. React treats components starting with lowercase letters as DOM tags. For example, represents an HTML div tag, but <Welcome /> represents a component and requires Welcome to be in scope.

11. A good rule of thumb is that if a part of your UI is used several times (Button, Panel, Avatar), or is complex enough on its own (App, FeedStory, Comment), it is a good candidate to be a reusable component.

12. Whether you declare a component as a function or a class, it must never modify its own props. Such functions are called "pure" because they do not attempt to change their inputs, and always return the same result for the same inputs.

13. All React components must act like pure functions with respect to their props.

14. In JavaScript classes we have get and set inside of classes (also objects).

```
// Getter
  get area() {
    return this.calcArea();
  }
  // Method
  calcArea() {
    return this.height * this.width;
  }
}
```

15. The static keyword defines a static method for a class. Static methods are called without instantiating their class and cannot be called through a class instance

16. Code within the class body's syntactic boundary is always executed in strict mode.

17. Class components should always call the base constructor with props.

18. State Declaration

```
this.state = {date: new Date()};
```

19. The componentDidMount() method runs after the component output has been rendered to the DOM. This is a good place to set up a timer

20. We will tear down the timer in the componentWillUnmount() lifecycle method. In this lifecycle method is called when the component is not being rendered in the DOM.

21. Do Not Modify State Directly

```
this.setState({comment: 'Hello'});
```

22. The only place where you can assign this.state is the constructor.

23. State Updates May Be Asynchronous. React may combine several setState into a single batch for improved performance. Because of this never use this.state and this.props for calculating the next state. To fix this issue we pass a callback function instead of an object.

```
// assuming this.state.count === 0
this.setState({count: this.state.count + 1});
this.setState({count: this.state.count + 1});
this.setState({count: this.state.count + 1});
// this.state.count === 1, not 3
```

React will combine all the setStates and execute them together. Now initially all will get the value of 0. So output is 1.

Instead of passing in an object to this.setState we can pass in a function and reliably get the value of the current state of our component.

```
this.setState((state, props) => ({
  counter: state.counter + props.increment
}));
```

We actually the expression which is an object.

```
const ac = () => (
    5
)
const ac = () => 5
```

Both of them are same.

24. When you call setState(), React merges the object you provide into the current state.

25. States can be passed as properties. However never ever pass your props into state.

26. <button onClick={activateLasers}>

27. Another difference is that you cannot return false to prevent default behaviour in React. We must call .preventDefault() to change the default behaviour.

28. In JavaScript, class methods are not bound by default.

29. Generally, if you refer to a method without () after it, such as onClick={this.handleClick}, you should bind that method.

30. this.handleClick = this.handleClick.bind(this); // in the constructor

31. <button onClick={(e) => this.handleClick(e)}>

32. We generally recommend binding in the constructor or using the class fields syntax

33. <button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>

34. <button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>

If we do not want to specify a bind in the constructor then we can do the above.

35. We can do some computations before the return of the render method.
36. Inline if && expression

```
{unreadMessages.length > 0 &&
  <h2>
    You have {unreadMessages.length} unread messages.
  </h2>
}
```

This occurs because true && expression evaluates to expression in JS.
false && expression always evaluates to false.

37. The user is {isLoggedIn ? 'currently' : 'not'} logged in.
38. If we do not want a component to render, then return null.
39. Rendering multiple components

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);

ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
```

```
);
```

40. When we create a list of components we need to specify a key

```
<li key={number.toString()}>
  {number}
</li>
```

41. Keys used within arrays should be unique among their siblings. However they don't need to be globally unique. We can use the same keys when we produce two different arrays:

42. Keys serve as a hint to React but they don't get passed to your components. If you need the same value in your component, pass it explicitly as a prop with a different name:

43. In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. In React, mutable state is typically kept in the state property of components, and only updated with setState(). We can combine the two by making the React state be the "single source of truth". Then the React component that renders a form also controls what happens in that form on subsequent user input. An input form element whose value is controlled by React in this way is called a "controlled component".

44. Handling inputs

```
handleChange(event) {
  this.setState({value: event.target.value});
}

handleSubmit(event) {
  alert('A name was submitted: ' + this.state.value);
  event.preventDefault();
}
```

```
<input type="text" value={this.state.value}
onChange={this.handleChange} />
```

45. With a controlled component, every state mutation will have an associated handler function. This makes it straightforward to modify or validate user input

46. You can pass an array into the value attribute, allowing you to select multiple options in a select tag:

```
<select multiple={true} value={['B', 'C']}>
```

47. To know if checkbox is selected or not use event.target.checked

48. When you need to handle multiple controlled input elements, you can add a name attribute to each element and let the handler function choose what to do based on the value of event.target.name.

49. Computed Property Names

```
let param = 'size'
let config = {
  [param]: 12,
  ['mobile' + param.charAt(0).toUpperCase() + param.slice(1)]: 4
}

console.log(config) // {size: 12, mobileSize: 4}
```

This is the use. Of computed property names. We can use computed property names to update the value of the state.

50. The only method you must define in a React.Component subclass is called render()

51. static getDerivedStateFromError(error). This lifecycle method will be invoked when an error is thrown by any of the child components.

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  render() {
```

```
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

52. 107

53. Shallow compare does check for equality. When comparing scalar values (numbers, strings) it compares their values. When comparing objects, it does not compare their's attributes - only their references are compared (e.g. "do they point to same object?).

54. <React.StrictMode> </React.StrictMode>

55. Use <React.Fragment> </ React.Fragment> if we do not want the extra div tag.

56. We can also use <> </>