Imports

```
import React, { useState } from 'react';
import './index.css';
```

- **`useState`:** This hook from React allows you to add state variables to functional components. State represents data that can change within a component, and React will automatically re-render the component whenever this state changes.
- **`./index.css`:** This line imports your custom CSS file (index.css) into the component. This CSS file will contain the styling for your quiz, defining the appearance of elements like the question container, buttons, text, etc.

2. `Question` Component

```
function Question({ questionText, options, correctAnswer,
selectedOption, onSelectOption }) {
```

- **Functional Component:** This is a stateless (or presentational) component written as a JavaScript function. It takes in data (props) and returns JSX (a syntax extension for JavaScript that allows you to write HTML-like code).
- **Destructuring Props:** The curly braces `{}` within the function's parameters are used to destructure the props object. This allows you to directly access the individual prop values without writing `props.questionText`, `props.options`, etc.

```
  return (
    <div>
      <p>{questionText}</p>
      {options.map((option) => (
        <label key={option}>
          <input
            type="radio"
            name={questionText}
            value={option}
            checked={selectedOption === option}
            onChange={() => onSelectOption(option)} // Call
the parent's handler
          />
          {option}
        </label>
      ))}
```

```
    </div>
  );
}
```

.

- **JSX Structure:** The component returns a `<div>` that contains:
  - A `<p>` tag to display the `questionText`.
  - A list of radio buttons created using the `map` function. The `map` function iterates over the `options` array, creating a `<label>` and `<input type="radio">` element for each option.
- **Radio Button Attributes:**
  - `key={option}`: Assigns a unique identifier to each radio button, which is crucial for React to efficiently update the list if the options change.
  - `name={questionText}`: Groups the radio buttons together, ensuring that only one option can be selected at a time for each question.
  - `value={option}`: Sets the value that will be submitted with the form (the answer choice).
  - `checked={selectedOption === option}`: If the current `selectedOption` (stored in the parent component) matches the current `option`, this radio button will be checked.
  - `onChange={() => onSelectOption(option)}`: This is an event handler. When a radio button is selected, it calls the `onSelectOption` function, which is a function passed down from the parent `Quiz` component to update the selected answer in its state.

3. `Quiz` Component

```
function Quiz() {
  // ... (useState and questions data)
}
```

.

- **State Variables:**
  - `score`: Keeps track of how many questions the user has answered correctly. Initialized to 0.
  - `currentQuestionIndex`: Keeps track of the index of the question currently being displayed. Initialized to 0, meaning the first question.
  - `selectedOption`: Stores the user's selected answer for the current question. Initialized to `null` since no answer is chosen initially.

```
  const handleNextQuestion = () => {
```

```
    if (questions[currentQuestionIndex].correctAnswer ===
selectedOption) {
      setScore(score + 1);
    }
    setCurrentQuestionIndex(currentQuestionIndex + 1);
    setSelectedOption(null); // Reset for next question
  };
```

- **Handling "Next" Button Click (`handleNextQuestion`):**
  - This function is called when the user clicks the "Next" button.
  - First it checks if the answer to the current question matches the `correctAnswer` stored in `questions`. If it does, it updates the `score`.
  - Next, it updates `currentQuestionIndex` to move to the next question in the `questions` array.
  - Finally, it resets `selectedOption` to `null` to clear the selection for the next question.

```
  return (
    <div>
      {currentQuestionIndex < questions.length ? (
        <>
          <Question

questionText={questions[currentQuestionIndex].questionText}

options={questions[currentQuestionIndex].options}

correctAnswer={questions[currentQuestionIndex].correctAnswe
r}
            selectedOption={selectedOption}
            onSelectOption={setSelectedOption}
          />
          <button onClick={handleNextQuestion}>Next</
button>
        </>
      ) : (
        <p>You scored {score} out of {questions.length}</p>
      )}
    </div>
  );}
```

- **Conditional Rendering:**
  - If there are still questions to be answered (`currentQuestionIndex` is less than the length of the `questions` array):
    - Render the `Question` component and pass the necessary props to it (question data, current selection, update function)
    - Render the "Next" button
  - If all questions have been answered (`currentQuestionIndex` is no longer less than the length of the `questions` array):
    - Render a message displaying the final score.