**1. React**

**React** — это библиотека для создания пользовательских интерфейсов. Она позволяет создавать компоненты, которые можно повторно использовать и комбинировать для создания сложных интерфейсов.

**2. JSX**

**JSX** (JavaScript XML) — это синтаксис, используемый в React для описания структуры пользовательского интерфейса. JSX позволяет писать HTML-подобный код внутри JavaScript. Это делает код более читабельным, но его нужно компилировать с помощью Babel.

Пример JSX:

const element = <h1>Hello, world!</h1>;

**3. Babel**

**Babel -** это инструмент транспиляции для JavaScript, который позволяет разработчикам использовать последние возможности языка JavaScript, даже если они не поддерживаются в текущей версии браузера или среды выполнения Node.js. Он позволяет использовать последние функции JavaScript, не беспокоясь о совместимости. **Транспиляция** — преобразование программы, написанной на одном языке программирования в качестве исходных данных, в эквивалентный код другой версии этого языка или в другой язык программирования того же уровня абстракции.

**4. SPA (Single Page Application)**

**Single-page application (SPA) — э**то веб-приложение, которое загружает только одну HTML-страницу и динамически обновляет ее содержимое без необходимости полной перезагрузки страницы при взаимодействии с пользователем. Это позволяет обеспечить более плавный пользовательский опыт, так как отсутствует необходимость перезагрузки страницы.

**5. Рендеринг**

**Рендеринг** в контексте React — это процесс отображения компонентов на экране. React использует виртуальный DOM для повышения производительности. Когда состояние компонента изменяется, React обновляет только измененные части реального DOM, вместо того чтобы перерисовывать всю страницу.

**6. Компоненты**

**Компоненты** — это строительные блоки React-приложений. Каждый компонент отвечает за отображение определенной части интерфейса. Компоненты могут быть функциональными или классовыми.

Пример функционального компонента:

const MyComponent = () => {

return <div>Hello!</div>;

};

**7. Пропсы**

**Пропсы** (свойства) — это параметры, которые передаются компонентам. Они позволяют передавать данные и функции между компонентами. Пропсы являются неизменяемыми (иммутабельными) внутри компонента.

Пример использования пропсов:

const Greeting = ({ name }) => {

return <h1>Hello, {name}!</h1>;

};

// Использование

<Greeting name="Alice" />

**8. Состояние**

**Состояние** (state) — это изменяемые данные, которые управляются внутри компонента. Состояние может изменяться в ответ на действия пользователя, например, нажатие кнопок.

Пример использования состояния с хуком useState:

import React, { useState } from 'react';

const Counter = () => {

const [count, setCount] = useState(0);

return (

<div>

<h1>{count}</h1>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

};

**9. Жизненные циклы компонента**

**Жизненные циклы компонента** — это этапы, через которые проходит компонент от создания до удаления. В классовых компонентах есть методы жизненного цикла, такие как componentDidMount, componentDidUpdate и componentWillUnmount. В функциональных компонентах можно использовать хуки, такие как useEffect, для управления побочными эффектами.

Пример использования useEffect:

import React, { useEffect } from 'react';

const MyComponent = () => {

useEffect(() => {

// Код, который выполнится после монтирования компонента

return () => {

// Код, который выполнится перед размонтированием компонента

};

}, []); // Пустой массив означает, что эффект выполнится только один раз

return <div>Hello!</div>;

};

**10. Принцип единственной ответственности**

**Принцип единственной ответственности** (Single Responsibility Principle) — это принцип программирования, согласно которому каждый компонент должен иметь одну конкретную задачу. Это делает код более чистым и легким в сопровождении.

**11. Односторонний поток данных**

**Односторонний поток данных** — это концепция, согласно которой данные в React передаются сверху вниз. Родительский компонент передает пропсы дочерним компонентам, и дочерние компоненты не могут изменять данные, полученные из родительского компонента.

**12. Иммутабельность**

**Иммутабельность** — это свойство, согласно которому данные не могут быть изменены после их создания. В React это важно, так как изменение состояния должно происходить с помощью функции обновления состояния, что позволяет React отслеживать изменения и эффективно обновлять интерфейс.

**13. Чистая функция**

**Чистая функция** — это функция, которая для одних и тех же входных данных всегда возвращает одни и те же выходные данные и не имеет побочных эффектов (например, не изменяет внешние состояния).

**14. Функциональная компонента**

**Функциональная компонента** — это компонента, написанная как функция. Она может использовать хуки для управления состоянием и побочными эффектами. Функциональные компоненты проще и легче в понимании, чем классовые компоненты.

**15. Хук**

**Хук** — это функция, которая позволяет вам "привязывать" состояние и другие возможности React к функциональным компонентам. Наиболее распространенные хуки — это useState и useEffect.

**16. useState**

**useState** — это хук, который позволяет добавлять состояние в функциональные компоненты. Он возвращает массив, где первый элемент — текущее состояние, а второй — функция для его обновления.

**17. useEffect**

**useEffect** — это хук, который позволяет выполнять побочные эффекты в функциональных компонентах, такие как запросы к API или подписки. Он выполняется после рендера компонента и может быть настроен на выполнение в зависимости от изменений в состоянии или пропсах.