1. **Компоненты (Components):**
   * **Определение:** Компоненты - это основные строительные блоки пользовательского интерфейса (UI) в веб-разработке. Компоненты могут быть простыми, такими как кнопка или текстовое поле, или более сложными, такими как форма или таблица.
   * **Пример (React):** В React, компонент может быть классовым компонентом, функциональным компонентом или хуком. Пример функционального компонента:

|  |
| --- |
| function MyComponent(props) {  return <div>{props.message}</div>;  } |

Эта функция — компонент, потому что она получает данные в одном объекте («пропсы») в качестве параметра и возвращает React-элемент. Мы будем называть такие компоненты «функциональными», так как они буквально являются функциями.

Когда React встречает подобный элемент, он собирает все JSX-атрибуты и дочерние элементы в один объект и передаёт их нашему компоненту. Этот объект называется «пропсы» (props).

Компоненты могут ссылаться на другие компоненты в возвращённом ими дереве. Это позволяет нам использовать одну и ту же абстракцию — компоненты — на любом уровне нашего приложения. Неважно, пишем ли мы кнопку, форму или целый экран: все они, как правило, представляют собой компоненты в React-приложениях.

Например, компонент App может отрендерить компонент Welcome несколько раз:

function Welcome(props) {

return <h1>Привет, {props.name}</h1>;

}

function App() {

return (

<div>

<Welcome name="Алиса" />

<Welcome name="Базилио" />

<Welcome name="Буратино" />

</div>

);

}

React-элементы — это составляющие блоки React-приложений. Их можно перепутать с более известной концепцией «компонентов», но в отличие от компонента, элемент описывает то, что вы хотите увидеть на экране. React-элементы иммутабельны.

1. **Пропсы (Props):**
   * **Определение:** Пропсы - это параметры, передаваемые компоненту для настройки его поведения или внешнего вида.
   * **Пример (React):**

|  |
| --- |
| function Greeting(props) {  return <div>Hello, {props.name}!</div>;  }  // Использование компонента с пропсами  <Greeting name="John" />; |

props (пропсы) — это входные данные React-компонентов, передаваемые от родительского компонента дочернему компоненту.

Помните, props предназначены только для чтения. Ни в каком случае их не следует изменять

1. **Стейт (State):**
   * **Определение:** Стейт представляет изменяемые данные, хранящиеся внутри компонента. Обновление стейта приводит к перерендерингу компонента.
   * **Пример (React):**

|  |
| --- |
| import React, { useState } from 'react';  function Counter() {  const [count, setCount] = useState(0);  return (  <div>  <p>Count: {count}</p>  <button onClick={() => setCount(count + 1)}>Increment</button>  </div>  );  } |

Компонент нуждается в state, когда данные в нём со временем изменяются. Например, компоненту Checkbox может понадобиться состояние isChecked, а компоненту NewsFeed необходимо отслеживать посты при помощи состояния fetchedPosts.

Самая большая разница между state и props состоит в том, что props передаются от родителя потомку, а state управляется самим компонентом. Компонент не может изменять props, но может изменять state.

Для каждой отдельной части изменяемых данных должен существовать только один компонент, который «управляет» изменением состояния. Не пытайтесь синхронизировать состояния двух разных компонентов. Вместо этого [поднимите оба этих состояния](https://ru.legacy.reactjs.org/docs/lifting-state-up.html) до ближайшего компонента-родителя и передайте через пропсы необходимым дочерним компонентам.

1. **Хуки (Hooks):**
   * **Определение:** Хуки в React - это функции, которые позволяют использовать состояние и другие возможности React в функциональных компонентах.
   * **Пример (React):**

|  |
| --- |
| import React, { useState, useEffect } from 'react';  function ExampleComponent() {  const [data, setData] = useState(null);  useEffect(() => {  // Логика для загрузки данных  // Вызов setData для обновления стейта  }, []); // Пустой массив означает, что эффект выполняется только при монтировании  return <div>{data}</div>;  } |

В этом примере, useState — это хук (определение хука дано ниже). Мы вызываем его, чтобы наделить наш функциональный компонент внутренним состоянием. React будет хранить это состояние между рендерами. Вызов useState возвращает массив с двумя элементами, который содержит: текущее значение состояния и функцию для его обновления. Эту функцию можно использовать где угодно, например, в обработчике событий. Она схожа с this.setState в классах, но не сливает новое и старое состояние вместе. Сравнение хука useState и this.state приводится на странице [Использование хука состояния](https://ru.legacy.reactjs.org/docs/hooks-state.html).

Единственный аргумент useState — это начальное состояние. В примере выше — это 0, так как наш счётчик начинается с нуля. Заметьте, что в отличие от this.state, в нашем случае состояние может, но не обязано, быть объектом. Исходное значение аргумента используется только при первом рендере.

1. **Redux:**
   * **Actions, Reducers, Dispatch:**
     + **Actions:** Объекты, описывающие события. Они отправляются в хранилище Redux для запуска изменений.
     + **Reducers:** Функции, которые обрабатывают события (actions) и возвращают новое состояние.
     + **Dispatch:** Метод, используемый для отправки actions в Redux store.
   * **Пример (Redux):**

|  |
| --- |
| // Action  const increment = () => ({  type: 'INCREMENT',  });  // Reducer  const counterReducer = (state = 0, action) => {  switch (action.type) {  case 'INCREMENT':  return state + 1;  default:  return state;  }  };  // Dispatch  store.dispatch(increment()); |

Эти концепции актуальны в контексте React и Redux, но они также могут использоваться в других фреймворках и библиотеках для разработки веб-приложений. Если у вас есть более конкретные вопросы или если вам нужно больше информации по какой-то из тем, дайте мне знать!