**React** — это JavaScript-библиотека для разработки пользовательского интерфейса. (декларативная)

**Основные характеристики:**

* **Компонентный подход.** Пользовательский интерфейс разбивается на небольшие, независимые и переиспользуемые компоненты.
* **Виртуальный DOM,** который используется для эффективного обновления только измененных частей интерфейса. Каждый раз, когда происходят изменения, React сравнивает виртуальный DOM с реальным, и обновляет только необходимые элементы.
* **Однонаправленный поток данных.** Данные передаются от родительских компонентов дочерним. Это обеспечивает предсказуемость и упрощает управление состоянием приложения.
* **JSX (JavaScript XML)** – расширение синтаксиса JavaScript, которое позволяет описывать структуру пользовательского интерфейса в виде XML-подобного кода прямо внутри JS. Это облегчает интеграцию HTML и JavaScript.
* **Реактивные обновления**: React обеспечивает автоматическое обновление пользовательского интерфейса при изменении состояния.

**Функции создания компонентов**

**React.createElement( type, [props], [...children]) –** создает и возвращает новый React-элемент определенного типа. Аргументом type может юыть строка, содержащая имя тега (например, ‘div’ или ‘span’), React-компонент (класс или функция), или React-фрагмент.

Код, написанный с использованием JSX, будет преобразован в React.createElement(). Обычно вы не будете вызывать React.createElement() напрямую, если вы используете JSX.

**render(element, container[, callback])** – устаревший метод, который рендерит указанный элемент в указанный контейнер DOM и возвращает ссылку на компонент.

Если элемент уже был ранее отрендерен в container, то повторный вызов произведет его обновление и изменит соответствующую часть DOM, чтобы она содержала последние изменения.

Callback вызывается после того, как компонент отрендерится.

**createRoot()** - это новый метод, добавленный в React версии 16.9, который используется для создания корневого компонента React, вместо render().

**JSX -** синтаксический сахар для функции React.createElement(component, props, ...children).



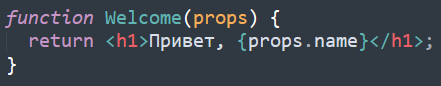
Скомпилируется в



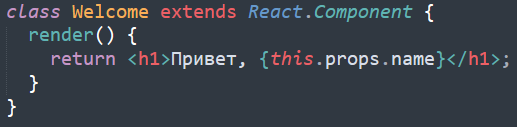
Значения *false*, *null*, *undefined* и *true* — валидные дочерние компоненты. Просто они не рендерятся.

**React-компоненты –** повторно используемые части кода, которые возвращают элементы для отображения на странице.

Компонент может быть **Функциональным**:



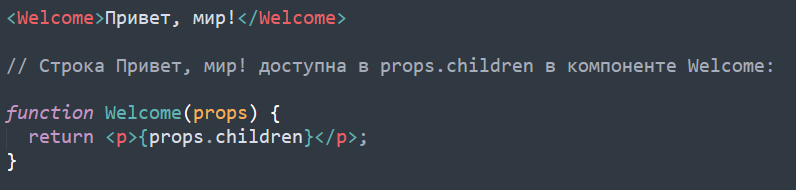
Либо **классовый**:



**Props (сокращение от "properties") –** это входные данные для React-компонентов, передаваемые от родительского компонента дочернему компоненту.

Props предназначены **только для чтения**. Ни в каком случае их не следует изменять. Если вам нужно поменять значение в ответ на пользовательский ввод или ответ сервера, используйте *state* (состояние).

**props.children –** контент между открывающимся и закрывающимся тегом компонента. Нпример:



**State –** объект который хранит информацию о состоянии компонента.

Изменение состояния происходит не напрямую, а через вызов метода **setState()**. Когда state изменяется, React перерисовывает компонент, чтобы отразить новое состояние.

Например, компоненту Checkbox может понадобиться состояние isChecked, а компоненту NewsFeed необходимо отслеживать посты при помощи состояния fetchedPosts.

Пример компонента счетчика с меняющимся состоянием:



**Методы жизненного цикла компонента**

**Монтирование**

При создании экземпляра компонента и его вставке в DOM, следующие методы вызываются в установленном порядке:

**constructor(), constructor(props)**

Вы можете не использовать конструктор в React-компоненте, если вы не определяете состояние или не привязываете методы.

Конструктор вызывается до того, как компонент будет примонтирован. В начале конструктора необходимо вызывать **super(props)**. Если этого не делать this.props не будет определен.

Конструкторы обычно используются для двух целей:

* Инициализация внутреннего состояния через присвоение объекта this.state.
* Привязка обработчиков событий к экземпляру.

Вы не должны вызывать setState() в конструкторе. Начальное состояние можно присвоить напрямую в **this.state.** Конструктор, единственное место, где можно напрямую изменить this.state. В остальных методах необходимо использовать this.setState().

Не используйте побочные эффекты или подписки в конструкторе. Вместо этого используйте *componentDidMount()*.

**static getDerivedStateFromProps() –** используется для обновления state на основе props. Вызывается перед каждым рендерингом компонента, как нового, так и при обновлении.

Принимает 2 параметра: props и state. Он должен вернуть объект, представляющий новое состояние объекта, или null если состояние не требуется обновлять.

**render() –** обязательный метод компонента React, который определяет, что будет отображаться на экране. Он возвращает React-элементы (JSX).

Метод *render()* вызывается автоматически при каждом обновлении компонента или его родительского компонента. Он должен быть чистой функцией без побочных эффектов, то есть он не должен изменять состояние компонента и не должен взаимодействовать напрямую с брузером.

Взаимодействовать с браузером необходимо в *componentDidMount()* или других методах жизненного цикла. Чистый render() делает компонент понятным.

render() может возвращать один из следующих вариантов:

* **Элемент React.** Обычно создается с помощью JSX.
* **Массив.** Возвращает несколько элементов из render().
* **Портал.** Позволяет рендерить дочерние элементы в DOM-узлы, которые находятся вне иерархии компонентов.

Порталы полезны когда вам нужно отобразить содержимое вне корневого узла вашего приложения, например, для создания модальных окон, всплывающих подсказок и т.п.

* **Строки и числа –** рендерит текстовые DOM-узлы.
* **Boolean или null –** ничего не рендерит. Обычно необходим для поддержки паттерна *return test && <Child />*, где test — логическое значение)