1. Компонент React – это класс, который наследуется от класса React.Component (как вы увидите позже, это не единственный способ создать компонент)
2. Функция render возвращает нечто (рассмотрим позже), что будет отрисовано в браузере. Класс-компонент без функции render существовать не может, это его интерфейс

**JSX**

JSX – это похожее на XML-разметку расширение для JavaScript, созданное специально для задач React. React из коробки поставляется с набором компонентов, которые полностью повторяют HTML. По большей части синтаксис и структура JSX и HTML совпадают, но есть некоторые важные различия:

1. Так как это похожий на XML синтаксис, одиночные теги в JSX должны быть закрыты: <hr />
2. Вместо атрибута class в JSX используется имя свойства в DOM: className, так как class — это зарезервированное слово для создания классов

**const** name **=** 'Eva';

**const** cname **=** 'container';

**const** vdom1 **=** <div>Hello, {name}</div>;

**const** vdom2 **=** <div>Hello, {name.repeat(3)}</div>;

**const** vdom3 **=** <div className={cname}>Hello!</div>;

Как видно, вставка (по сути – интерполяция) происходит за счёт использования фигурных скобок, причём внутри них может быть любое выражение. Эта схема работает одинаково как для содержимого тегов, так и для атрибутов.

export default ({ title, text }) => {

  if (!title && !text) {

    return null;

  }

  const titleDom = title && <h4 className="card-title">{title}</h4>;

  const textDom = text && <p className="card-text">{text}</p>;

  return (

    <div className="card">

      <div className="card-body">

        {titleDom}

        {textDom}

      </div>

    </div>

  );

};

ПРОПСЫ!!!

import React from 'react';

// BEGIN (write your solution here)

export default class Card extends React.Component {

  render() {

    const { title, text } = this.props;

    return (

      <div className="card">

        <div className="card-body">

          <h4 className="card-title">{title}</h4>

          <p className="card-text">{text}</p>

        </div>

      </div>

    );

  }

}

Card.defaultProps = {

  text: 'text',

  title: 'title'

};

class List extends React.Component {

render() {

const { data } = this.props

return (

<ul>

{data.map(item => <li>{item.name}</li>)}

</ul>

);

}

}

const items = [

{ name: 'first' },

{ name: 'second' },

{ name: 'three'}

];

const mountNode = document.getElementById('react-root');

const root = ReactDOM.createRoot(mountNode);

root.render(<List data={items} />);

## Использование React.Fragment

class Columns extends React.Component {

render() {

return (

<React.Fragment> <td>Привет</td>

<td>Мир</td>

</React.Fragment> );

}

}

Результатом будет правильный вывод <Table />:

<table>

<tr>

<td>Привет</td>

<td>Мир</td>

</tr>

</table>

### Сокращённая запись

Существует сокращённая запись объявления фрагментов. Она выглядит как пустые теги:

class Columns extends React.Component {

render() {

return (

<> <td>Привет</td>

<td>Мир</td>

</> );

}

}

## Стили

Совсем по-другому работает атрибут style. Если в HTML это обычная строка, то в JSX только объект.

class Component extends React.Component {

render() {

const divStyle = {

color: 'blue',

fontSize: '50px',

};

return <div style={divStyle}>Hello World!</div>;

}

}

export default class Definitions extends React.Component {

  render() {

    const { data } = this.props;

    if (data.length === 0) {

      return null;

    }

    const tags = data.map(({ dt, dd, id }) => (

      <React.Fragment key={id}>

        <dt>{dt}</dt>

        <dd>{dd}</dd>

      </React.Fragment>

    ));

    return (

      <dl>

        {tags}

      </dl>

    );

  }

}

ТОЖЕ САМОЕ

export default class Definitions extends React.Component {

  renderList() {

    const { data } = this.props;

    const tags = data.map(({ dt, dd, id }) => (

      <React.Fragment key={id}>

        <dt>{dt}</dt>

        <dd>{dd}</dd>

      </React.Fragment>

    ));

    return tags;

  }

  render() {

    const { data } = this.props;

    if (data.length === 0) {

      return null;

    }

    return (

      <dl>

        {this.renderList()}

      </dl>

    );

  }

}

Classnames

<https://github.com/JedWatson/classnames>

import cn from 'classnames';

import React from 'react';

// BEGIN (write your solution here)

 export default class Alert extends React.Component {

  render () {

    const { type, text } = this.props;

    const alertClass = cn('alert', `alert-${type}`);

    return (

      <div className={alertClass} role="alert">{text}</div>

    )

  }

};

CHILDREN

class Alert extends React.Component {

render() {

const { children } = this.props;

return (<div className="alert alert-primary">

{children}

</div>);

}

}

const vdom = <Alert>

<p>Aww yeah, you successfully read this important alert message. This example text is going to run a bit longer so that you can see how spacing within an alert works with this kind of content.</p>

<hr />

<p class="mb-0">Whenever you need to, be sure to use margin utilities to keep things nice and tidy.</p>

</Alert>;

const mountNode = document.getElementById('react-root');

const root = ReactDOM.createRoot(mountNode);

root.render(vdom);

Обратите внимание на то, что компонент стал использоваться как парный тег в JSX:

**const** vdom **=** (

<**Alert**>

<p>Whenever you need to, be sure to use margin utilities to keep things nice and tidy.</p>

</**Alert**>

);

Все, что находится между открывающим и закрывающим тегом, попадает внутрь пропса *children*.

**Композиция компонентов**

Иерархия компонентов во многом закладывает структуру приложения. То, как вы распределите приложение на компоненты определит кодовую базу вашего проекта.

Есть несколько советов как разбивать приложение на компоненты:

* выделите части интерфейса в приложении, которые имеют осмысленный функционал. Например: навигационная панель, боковое меню, окно чата и т.д. Каждую такую часть приложения можно выделить в отдельный компонент. Такое разбиение интерфейса на состовляющие части заложит основу компонентной базы
* если какие-то элементы повторяются, то их тоже лучше выделить в отдельный компонент для переиспользования. Например это могут быть Button и InputField
* рассматривайте компоненты как чистые функции. Пропсы, как и параметры в чистых функциях, не должны изменяться в компонентах, в React это важное правило

export default class ListGroup extends React.Component {

  render() {

    const { children } = this.props;

    return (

      <ul className="list-group">

        {React.Children.map(children, (child) => <li className="list-group-item">{child}</li>)}

      </ul>

    )

  }

}