Сейчас мы с вами более глубоко окунемся в работу с JSX: разберем некоторые его особенности и ограничения, а также научимся делать вставки переменных, работать с условиями и циклами.

На чем мы закончили

Давайте вспомним, на чем мы закончили предыдущий урок:

```
<div id="content"></div>

                    class App extends React.Component {

                                render() {

                              return <div>текст</div>;

                                    }

                                 }


                            ReactDOM.render(

                                  <App />,

                          document.getElementById('content')
);
```

Результатом этого кода будет следующий HTML:

```
                          <div id="content">

                             <div>текст</div>
</div>
```

Особенности метода render

Давайте теперь поиграемся с JSX. Модифицируем результат работы метода render. Можно разбить наш див на несколько строк:

```
                    class App extends React.Component {

                                render() {

                                  return <div>
```

```
            текст
        </div>;
        }
}
```

Однако, просто снести вниз нельзя - это выдаст ошибку:

```
class App extends React.Component {
    render() {

    //Такое выдаст ошибку:
        return
            <div>
                текст
            </div>;
        }
}
```

Для решения этой проблемы можно поставить круглые скобки:

```
class App extends React.Component {
    render() {

    //Это будет работать:
        return (
            <div>
                текст
            </div>
        );
        }
}
```

Несколько тегов в return

Результатом рендеренга не может быть несколько тегов:

```
class App extends React.Component {

render() {


//Такое выдаст ошибку:

return (

<p>текст 1</p>

<p>текст 2</p>

);

}

}
```

Несколько тегов обязательно нужно обернуть в какой-то общий тег. Давайте обернем наши абзацы в общий тег div:

```
class App extends React.Component {

render() {


//Это будет работать:

return (

<div>

<p>текст 1</p>

<p>текст 2</p>

</div>

);

}

}
```

Переменные

Как уже упоминалось ранее, кусочки HTML кода можно записывать в переменные. Давайте сделаем это внутри метода render: запишем кусочек текста в переменную text, а затем напишем эту переменную после return:

```
class App extends React.Component {
    render() {
        const text = <div>текст</div>;
        return text;
    }
}
```

Результатом работы этого кода будет следующее:

```
<div>
    текст
</div>
```

Вставка значений переменных

Пусть у нас есть переменная с каким-то текстом, например, переменная с именем str. Мы можем вставить значение этой переменной в любой кусочек HTML кода написав ее в фигурных скобках вот так: {str} . В этом случае при рендеринге вместо переменной в фигурных скобках вставится ее значение:

```
class App extends React.Component {
    render() {
        const str = 'текст';

        return (
            <div>
                {str}
            </div>
        );
```

```
    }
}
```

Результатом работы этого кода будет следующее:

```
<div>
  текст
</div>
```

В переменной вместо текста может хранится кусочек HTML кода - это не имеет значения, все будет работать:

```
class App extends React.Component {
  render() {
    const str = <p>текст</p>;

    return (
      <div>
        {str}
      </div>
    );
  }
}
```

Результатом работы этого кода будет следующее:

```
<div>
<p>текст</p>
</div>
```

Учтите, что несколько тегов, хранящихся в переменной, обязательно нужно обернуть в какой-то общий тег. То есть вот так не будет работать:

```
class App extends React.Component {
  render() {
```

```
const list = <p>1</p><p>2</p><p>3</p>;


return (
<div>
{list}
</div>
);
}
}
```

А вот так будет, так как абзацы обернуты в один тег div:

```
class App extends React.Component {
render() {
const list = <div><p>1</p><p>2</p><p>3</p></div>;


return (
<div>
{list}
</div>
);
}
}
```

Выполнение JavaScript

Итак, внутри кусочков HTML кода можно делать вставки переменных. Пусть у нас есть переменная num. Давайте вставим в определенное место содержимое этой переменной:

```
class App extends React.Component {
render() {
const num = 3;
```

```
        const text = <div>текст {num}</div>;

        return text;

        }
}
```

Результатом работы этого кода будет следующее:

```
<div id="content">

    <div>текст 3</div>

</div>
```

Внутри фигурных скобок можно не только вставлять переменные, но и выполнять произвольный JavaScript код. Давайте, к примеру, в момент вставки прибавим к переменной num единицу:

```
class App extends React.Component {

        render() {

         const num = 3;

    const text = <div>текст {num + 1}</div>;

         return text;

        }
}
```

Результатом работы этого кода будет следующее:

```
<div id="content">

    <div>текст 4</div>

</div>
```

Пусть у нас теперь есть две переменные: num1 и num2. Давайте выведем на экран сумму этих переменных:

```
class App extends React.Component {

        render() {

         const num1 = 1;
```

```
        const num2 = 2;
    const text = <div>текст {num1 + num2}</div>;
        return text;
        }
}
```

Результатом работы этого кода будет следующее:

```
<div id="content">
    <div>текст 3</div>
</div>
```

Работа с атрибутами тегов

Вставку переменных можно делать не только в тексты тегов, но и в атрибуты. При этом *кавычки от атрибутов не ставятся*:

```
class App extends React.Component {
        render() {
        const str = 'elem';
    const text = <div id={str}>текст</div>;
        return text;
        }
}
```

Результатом работы этого кода будет следующее:

```
<div id="content">
    <div id="elem">текст</div>
</div>
```

Такое работает для всех атрибутов, но есть исключение. Вместо атрибута class следует писать атрибут className:

```
class App extends React.Component {
```

```
    render() {
      const str = 'elem';
      const text = <div className={str}>текст</div>;
      return text;
    }
}
```

Результатом работы этого кода будет следующее:

```
<div id="content">
  <div class="elem">текст</div>
</div>
```

Учтите, что даже если вы пишите напрямую имя класса в атрибуте class, все равно следует писать className:

```
class App extends React.Component {
    render() {
      const text = <div className="content">текст</div>;
      return text;
    }
}
```

Работа с CSS

С помощью атрибутов можно также добавлять CSS стили элементам.

Для этого в атрибут style передается объект с названиями CSS свойств и их значениями.

Сделаем, для примера, наш див красного цвета и размером шрифта в 30px:

```
class App extends React.Component {
    render() {
      const css = {color: 'red', fontSize: '30px'};
      const text = <div style={css}>текст</div>;
```

```
        return text;
    }
}
```

Можно объект написать прямо в атрибуте - в этом случае нам нужны две фигурные скобки: первая от JSX вставки, а вторая - от объекта:

```
class App extends React.Component {
    render() {
        const text = <div style={ {color: 'red', fontSize: '30px'} }>
            текст
        </div>;
        return text;
    }
}
```

В этом объекте можно также использовать переменные. Пусть например в переменной value хранится цвет. Давайте свойство color установим в значение из этой переменной:

```
class App extends React.Component {
    render() {
        const value = 'red';
        const text = <div style={ {color: value} }>
            текст
        </div>;
        return text;
    }
}
```

Условия

При работе с кусочками HTML можно применять условия if. Давайте сделаем так, чтобы в зависимости от содержимого переменной showText на экран вывелся один или другой текст:

```
class App extends React.Component {

render() {

const text;

const showText = true;


if (showText) {

text = <div>текст1</div>;

} else {

text = <div>текст2</div>;

}


return text;

}

}
```

Обратите внимание на то, что константу text нужно определить над ифом, иначе она не будет видна снаружи этого ифа.

Получится, что если переменная showText имеет значение true, то будет виден один текст, а если false - то другой.

Циклы

Кусочки HTML кода можно делать в цикле. Обычно для этого используется цикл map. Пример: пусть у нас есть массив arr, давайте каждый элемент этого массива обернем в тег li и сохраним этот набор li-шек в переменную list:

```
const arr = [1, 2, 3, 4, 5];
```

```
const list = arr.map(function(item, index) {
return <li>{item}</li>;
});
```

**В list запишется текст 1
2
3
4
5 - и это будет работать ни смотря на то, что эти лишки не обернуты в общий тег. То есть в цикле такое можно делать, а напрямую так сразу написать в переменную - нет.**

Давайте добавим метод render:

```
class App extends React.Component {
render() {
const arr = [1, 2, 3, 4, 5];

const list = arr.map(function(item, index) {
return <li>{item}</li>;
});

return <ul>
{list}
</ul>
}
}
```

Результатом работы этого кода будет следующее:

```
<ul>
<li>1</li>
<li>2</li>
<li>3</li>
<li>4</li>
<li>5</li>
```

```
</ul>
```

Проблема с ключами

В предыдущем примере мы формировали наши лишки в цикле, вот так:

```
const list = arr.map(function(item, index) {
    return <li>{item}</li>;
});
```

Это будет работать, однако, если заглянуть в консоль - мы увидим ошибку: *Warning: Each child in an array or iterator should have a unique "key" prop*. В данном случае React требует, чтобы каждой лишке мы дали уникальный номер, чтобы реакту было проще с этими лишками работать в дальнейшем.

Этот номер добавляется с помощью атрибута key. Чаще всего (но не обязательно) в качестве номера используется номер элемента в массиве. В нашем случае этот номер хранится в переменной index и значит исправленная строка будет выглядеть вот так: <li key={index}>{item}.

Давайте запустим - ошибка из консоли исчезнет:

```
class App extends React.Component {
    render() {
        const arr = [1, 2, 3, 4, 5];

        const list = arr.map(function(item, index) {
            return <li key={index}>{item}</li>;
        });

        return <ul>
            {list}
        </ul>
    }
}
```

Еще раз: этот атрибут key имеет служебное значение для реакта, более глубоко вы поймете этот момент в следующих уроках. Пока просто знайте: если вы видите такую ошибку - добавьте атрибут key с уникальным для каждого тега значением и проблема исчезнет.

Ключ key должен быть уникальным только внутри этого цикла, в другом цикле значения key могут совпадать со значениями из другого цикла.

Методы класса

Сейчас у нас вся работа происходит в классе App. Это обычный класс JavaScript, соответственно в нем мы можем делать какие-то свои методы. Давайте, к примеру, сделаем метод getText(), который будет возвращать какой-то текст:

```
class App extends React.Component {

getText() {

return 'текст';

}

}
```

Учтите, что метод render обязателен в классе, иначе React выкинет ошибку. В простых примерах я этот метод буду опускать, как это сделано здесь, чтобы он не отвлекал ваше внимание.

Вызовем наш метод getText в методе render и запишем результат работы нашего метода в переменную text. Напоминаю, чтобы вызвать метод, к нему следует обратиться через this, вот так: this.getText(). Смотрите пример:

```
class App extends React.Component {

getText() {

return 'текст';

}


render() {

const text = this.getText();
```

```
        return <div>{text}</div>;
    }
}
```

Можно не использовать промежуточную переменную text, а сделать return напрямую к результату работы нашего метода this.getText():

```
class App extends React.Component {
    getText() {
        return 'текст';
    }

    render() {
        return <div>{ this.getText() }</div>;
    }
}
```

Наш метод, конечно же, может возвращать не просто текст, а кусочек HTML кода:

```
class App extends React.Component {
    getText() {
        return <div>текст</div>;
    }

    render() {
        const text = this.getText();
        return text;
    }
}
```