Сейчас мы с вами будем разбираться с компонентами в фреймворке React. Компоненты представляют собой способ удобно реализовать ваш скрипт, разбить его на независимые блоки, каждый из которых имеет свою зону ответственности.

Компоненты

Взглянем на любой сайт. Он состоит из набора независимых блоков: хедер, сайдбары, футер, контент. Можно сказать, что эти блоки и есть компоненты в том смысле, в котором подразумевается в React.

Если посмотреть на тот же хедер, что в нем можно выделить блок с логотипом, блок контактов, блок с меню и так далее. То есть компоненты могут состоять из других подкомпонентов и так далее.

Аналогичным образом дело обстоит в React - сайт строится из набора компонентов, которые в свою очередь могут содержать другие компонеты и так далее.

Каждый компонент представляет собой класс. Обычно разработка начинается с главного компонента **App**, который содержит в себе остальные.

У каждого компонента могут быть свои стейты и каждый компонент имеет метод **render**, который возвращает результат работы этого компонента.

Давайте для примера сделаем компонент **User**, в стейте которого записаны имя и фамилия юзера, а результатом рендеринга является абзац с этим данными:

```
class User extends React.Component {
	constructor() {
		super();
		this.state = {name: 'Иван', surname: 'Иванов'};
	}

	render() {
		return <p>
			имя: {this.state.name}, фамилия: {this.state.surname}
		</p>;
	}
}
```

Давайте теперь подключим наш компонент **User** внутри компонента **App**. Это делается так: в то место, где должен появится результат работы нашего компонента следует вставить тег **<User />**, где **User** - имя нашего класса:

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

Соберем все вместе и запустим:

```
class User extends React.Component {
	constructor() {
		super();
		this.state = {name: 'Иван', surname: 'Иванов'};
	}

	render() {
		return <p>
			имя: {this.state.name}, фамилия: {this.state.surname}
		</p>;
	}
}

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

Результатом работы этого кода будет следующее:

```
<div>
	<p>
		имя: Иван, фамилия: Иванов
	</p>
</div>
```

Пропсы

Давайте теперь сделаем так, чтобы имя и фамилия нашего юзера хранились не в самом компоненте, а получались им извне.

Для этого при подключении компонента **<User />** необходимо добавить атрибуты **name** и **surname**, вот так:

```
class App extends React.Component {
	render() {
		return <div>
			<User name="Иван" surname="Иванов" />
		</div>;
	}
}
```

Значения добавленных атрибутов становятся доступны внутри нашего класса вот так: с помощью **this.props.name** мы можем получить имя, а с помощью **this.props.surname** - фамилию:

```
class User extends React.Component {
	constructor() {
		super();
	}

	render() {
		return <p>
			имя: {this.props.name}, фамилия: {this.props.surname}
		</p>;
	}
}
```

Таким образом, с помощью атрибутов можно передавать данные из родительского компонента в дочерний.

Конструкцию **this.props** в обиходе называют **пропсом** (по-русски переводится как *свойство*).

Получается, что у любого компонента есть стейты и пропсы. **Стейты** - это данные самого компонента, а **пропсы** - данные, переданные из родительского компонента.

Давайте соберем все вместе и запустим:

```
class App extends React.Component {
	render() {
		return <div>
			<User name="Иван" surname="Иванов" />
		</div>;
	}
}

class User extends React.Component {
	constructor(props) {
```

```
            super(props);
        }

        render() {
            return <p>
                имя: {this.props.name}, фамилия: {this.props.surname}
            </p>;
        }
}
```

Результатом работы этого кода будет следующее:

```
<div>
        <p>
            имя: Иван, фамилия: Иванов
        </p>
</div>
```

Преимущества компонентов

Преимущества компонентов проявляются, когда мы хотим вывести на экран не одного, а нескольких юзеров.

В этом случае один компонент для отображения многих юзеров удобнее: если нам нужно будет подправить то, как мы хотим отображать нашего юзера, - это удобнее будет сделать в отдельном компоненте, *не затрагивая* то, откуда берутся данные для нашего юзера.

То есть получится разделение: отображение данных отдельно от получения этих данных. Каждый компонент просто получает какие-то данные снаружи и отображает их в в нужно нам виде. А вот снаружи компонента данные есть, но мы их никак не отображаем - за это и отвечает компонент.

Давайте реализуем это: с помощью нашего компонента **User** выведем на экран несколько юзеров:

```
class User extends React.Component {
        constructor() {
            super();
        }

        render() {
            return <div>
                имя: {this.props.name}, фамилия: {this.props.surname}
            </div>;
        }
```

```
}

class App extends React.Component {
	constructor() {
		super();
	}

	//Выведем несколько юзеров:
	render() {
		return <div>
			<User name="Коля" surname="Иванов" />
			<User name="Вася" surname="Петров" />
			<User name="Петя" surname="Сидоров" />
		</div>;
	}
}
```

Результатом работы этого кода будет следующее:

```
<div>
	<p>
		имя: Коля, фамилия: Иванов
	</p>
	<p>
		имя: Вася, фамилия: Петров
	</p>
	<p>
		имя: Петя, фамилия: Сидоров
	</p>
</div>
```

Пусть теперь мы решили поредактировать способ отображения юзера и сделать так, чтобы имя и фамилия были в отдельном абзаце. В этом случае правки придется делать только в одном месте - в коде компонента **User**.

Давайте сделаем это:

```
class User extends React.Component {
	constructor() {
		super();
	}

	//Поменяем отображение отдельного юзера:
	render() {
		return <div>
			<p>имя: {this.props.name}</p>
			<p>фамилия: {this.props.surname}</p>
		</div>;
	}
}
```

```
class App extends React.Component {
	constructor() {
		super();
	}

	render() {
		return <div>
			<User name="Коля" surname="Иванов" />
			<User name="Вася" surname="Петров" />
			<User name="Петя" surname="Сидоров" />
		</div>;
	}
}
```

Запустите этот код и вы увидите, что отображение юзеров поменялось - теперь и имя, и фамилия расположены в отдельных абзацах.

Добавим стейты

Пусть теперь данные юзеров хранятся в стейте компонента **App**. Поправим наш код с соответствии с этим:

```
class User extends React.Component {
	constructor() {
		super();
	}

	render() {
		return <div>
			<p>имя: {this.props.name}</p>
			<p>фамилия: {this.props.surname}</p>
		</div>;
	}
}

class App extends React.Component {
	constructor() {
		super();

		//Храним данные в стейте:
		this.state = {
			user1: {name: 'Коля', surname: 'Иванов'},
			user2: {name: 'Вася', surname: 'Петров'},
			user3: {name: 'Петя', surname: 'Сидоров'},
		};
	}

	render() {
		return <div>
```

```
                  <User name={this.state.user1.name}
surname={this.state.user1.surname} />
                  <User name={this.state.user2.name}
surname={this.state.user3.surname} />
                  <User name={this.state.user3.name}
surname={this.state.user3.surname} />
            </div>;
      }
}
```

Если вы запустите этот код, то визуально ничего не поменяется, но данные будут браться уже из стейта компонента **App**.

Компоненты в цикле

В предыдущем примере мы использовали несколько экземпляров одного компонента, вот так:

```
class App extends React.Component {

      ...

      render() {
            return <div>
                  <User name={this.state.user1.name}
surname={this.state.user1.surname} />
                  <User name={this.state.user2.name}
surname={this.state.user3.surname} />
                  <User name={this.state.user3.name}
surname={this.state.user3.surname} />
            </div>;
      }
}
```

Давайте теперь сделаем то же самое, но так, чтобы за нас это сделал цикл.

Немного изменим стейт с массивом юзеров, вот так:

```
class App extends React.Component {
      constructor() {
            super();

            this.state = {
                  users: [
                        {name: 'Коля', surname: 'Иванов'},
                        {name: 'Вася', surname: 'Петров'},
                        {name: 'Петя', surname: 'Сидоров'},
                  ]
            };
```

```
	}
}
```

Давайте теперь сделаем цикл, который переберет наш массив с юзерами из **this.state.users** и сделает несколько экземпляров компонента **User**:

```
class App extends React.Component {

	...

	render() {
		const users = this.state.users.map((item, index) => {
			return <User
				key={index}
				name={item.name}
				surname={item.surname}
			/>;
		});

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

Обратите внимание на атрибут **key** - здесь он также обязателен для правильной внутренней работы React.

Давайте соберем все вместе и запустим - получим вывод компонентов в цикле:

```
class User extends React.Component {
	constructor() {
		super();
	}

	render() {
		return <div>
			<p>имя: {this.props.name}</p>
			<p>фамилия: {this.props.surname}</p>
		</div>;
	}
}

class App extends React.Component {
	constructor() {
		super();

		this.state = {
			users: [
				{name: 'Коля', surname: 'Иванов'},
```

```
                        {name: 'Вася', surname: 'Петров'},
                        {name: 'Петя', surname: 'Сидоров'},
                    ]
                };
        }

        render() {
            const users = this.state.users.map((item, index) => {
                return <User
                    key={index}
                    name={item.name}
                    surname={item.surname}
                />;
            });

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

Если вы запустите этот код, то визуально ничего не поменяется, но компоненты будут формироваться с помощью цикла.

Выводы

Компоненты позволяют разделить данные и способ их отображения на экране. Это удобно и позволяет упростить дальнейшую поддержку нашего кода.

Может быть, вам пока далеко не очевидно, что компоненты - это удобно. Не переживайте, просто научитесь ими пользоваться - удобства вы поймете только на практике, когда выполните несколько проектов.