



Лекция 2 - Props



Передача Props компоненту

React-компоненты используют **props** для взаимодействия друг с другом. Каждый родительский компонент может передавать некоторую информацию своим дочерним компонентам через **props**. **Props** могут напоминать **HTML**-атрибуты, но через них можно передавать любые значения JavaScript, включая объекты, массивы и функции.



Распространённые props

Свойства (**props**) — это информация, которую вы передаёте в JSX-тег.

Например, `className`, `src`, `alt`, `width` и `height` — это некоторые из **props**, которые можно передать в тег ``.

```
function Avatar() {  
  return (  
      
  );  
}  
  
export default function Profile() {  
  return (  
    <Avatar />  
  );  
}
```



Передача props в дочерний компонент

В этом коде компонент **Profile** не передаёт никаких **props** своему дочернему компоненту **Avatar**:

```
export default function Profile() {  
  return (  
    <Avatar />  
  );  
}
```



Шаг 1: Передача props в дочерний компонент

Сначала нужно передать некоторые **props** в **Avatar**.

Например, передадим два **props**: **person** (объект) и **size** (число):

```
export default function Profile() {  
  return (  
    <Avatar  
      person={{ name: 'Lin Lanying', imageId: '1bX5QH6' }}  
      size={100}  
    />  
  );  
}
```



Шаг 2: Чтение props внутри дочернего компонента

Вы можете прочитать эти **props**, указав их имена (**person**, **size**) через запятую внутри фигурных скобок (`{ ... }`) сразу после объявления функции

Avatar.

Это позволит использовать их внутри кода **Avatar**, так же, как обычные переменные.

```
function Avatar({ person, size }) {  
  // person and size are available here  
}
```



Шаг 2: Чтение props внутри дочернего компонента

Добавьте немного логики в **Avatar**, которая будет использовать **person** и **size** для рендеринга — и всё готово.


Теперь вы можете настраивать **Avatar**, чтобы он отображался по-разному в зависимости от переданных **props**. Попробуйте менять значения!

```
import { getImageUrl } from './utils.js';

function Avatar({ person, size }) {
  return (
    <img
      className="avatar"
      src={getImageUrl(person)}
      alt={person.name}
      width={size}
      height={size}
    />
  );
}
```

```
export default function Profile() {  
  return (  
    <div>  
      <Avatar  
        size={100}  
        person={{  
          name: 'Katsuko Saruhashi',  
          imageId: 'Yfe0qp2'  
        }}  
      />  
      <Avatar  
        size={80}  
        person={{  
          name: 'Aklilu Lemma',  
          imageId: 'OKS67lh'  
        }}  
      />  
      <Avatar  
        size={50}  
        person={{  
          name: 'Lin Lanying',  
          imageId: '1bX5QH6'  
        }}  
      />  
    </div>  
  );  
}
```





Указание значения по умолчанию для prop

Если вы хотите задать **значение по умолчанию** для пропа, чтобы использовать его в случае, когда значение не указано, это можно сделать через **деструктуризацию**, поставив знак `=` и значение сразу после параметра:

```
function Avatar({ person, size = 100 }) {  
  // ...  
}
```

Перенаправление props с помощью синтаксиса spread в JSX

```
function Profile({ person, size, isSepia, thickBorder }) {  
  return (  
    <div className="card">  
      <Avatar  
        person={person}  
        size={size}  
        isSepia={isSepia}  
        thickBorder={thickBorder}  
      />  
    </div>  
  );  
}
```

```
function Profile(props) {  
  return (  
    <div className="card">  
      <Avatar {...props} />  
    </div>  
  );  
}
```



Передача JSX через children

Когда вы вкладываете содержимое внутрь JSX-тега, родительский компонент получает это содержимое в пропе с названием **children**.

Например, компонент **Card** ниже получит проп **children**, равный `<Avatar />`, и отрендерит его внутри обёртки `<div>`.

```
import Avatar from './Avatar.js';

function Card({ children }) {
  return (
    <div className="card">
      {children}
    </div>
  );
}

export default function Profile() {
  return (
    <Card>
      <Avatar
        size={100}
        person={{
          name: 'Katsuko Saruhashi',
          imageId: 'Yfe0qp2'
        }}
      />
    </Card>
  );
}
```



Условный рендеринг (Conditional Rendering)

Обратите внимание, что у некоторых компонентов **Item** проп **isPacked** установлен в **true**, а не в **false**.

Вы хотите добавить галочку (✓) к упакованным элементам, если **isPacked={true}**.

Это можно записать с помощью конструкции **if/else**, например так:

```
if (isPacked) {  
  return <li className="item">{name} ✓</li>;  
}  
return <li className="item">{name}</li>;
```



Условный возврат null (ничего не рендерить)

В некоторых ситуациях вам может не понадобится рендерить что-либо вообще.

Например, представьте, что вы вовсе не хотите показывать упакованные вещи.


Компонент всегда должен что-то вернуть, в этом случае можно вернуть **null**:

```
if (isPacked) {  
  return null;  
}  
return <li className="item">{name}</li>;
```



Условное включение JSX (Conditionally including JSX)

Обе ветки условия возвращают JSX `<li className="item">...`

```
if (isPacked) {  
  return <li className="item">{name}  </li>;  
}  
return <li className="item">{name}</li>;
```



Условный (тернарный) оператор (?:)

Хотя такое дублирование и не является вредным, оно может усложнить поддержку кода.

А что если вы захотите изменить `className`? Вам придётся делать это в двух местах!

В такой ситуации можно условно включить небольшую часть JSX, чтобы сделать код более **DRY** (Don't Repeat Yourself — «не повторяйся»).

```
if (isPacked) {  
  return <li className="item">{name} ✓</li>;  
}  
return <li className="item">{name}</li>;
```

```
return (  
  <li className="item">  
    {isPacked ? name + ' ✓' : name}  
  </li>  
)
```

Логический оператор AND (&&)

Ещё один распространённый приём, с которым вы столкнётесь, — это логический оператор **AND (&&)** в JavaScript.

Внутри React-компонентов он часто используется, когда нужно отрендерить какой-то JSX, если условие истинно, или не рендерить ничего в противном случае.

С помощью **&&** можно условно отобразить галочку только тогда, когда `isPacked === true`:


```
return (  
  <li className="item">  
    {name} {isPacked && '✓'}  
  </li>  
);
```




Условное присваивание JSX переменной

Когда сокращённые приёмы мешают писать простой и понятный код, попробуйте использовать оператор `if` и переменную.

Переменные, объявленные с помощью `let`, можно переназначать, поэтому начните с указания содержимого по умолчанию, которое вы хотите отобразить — например, имени:

```
if (isPacked) {  
  itemContent = name + " ";  
}
```

```
<li className="item">  
  {itemContent}  
</li>
```



Рендеринг списков (Rendering Lists)

Часто возникает необходимость отображать несколько похожих компонентов из набора данных.

Для работы с массивами данных можно использовать методы массива JavaScript.

На этой странице вы будете применять **filter()** и **map()** в React, чтобы отфильтровать и преобразовать массив данных в массив компонентов.



Рендеринг данных из массивов (Rendering data from arrays)

Допустим, у вас есть список данных (контента).

```
<ul>
  <li>Creola Katherine Johnson: mathematician</li>
  <li>Mario José Molina-Pasquel Henríquez: chemist</li>
  <li>Mohammad Abdus Salam: physicist</li>
  <li>Percy Lavon Julian: chemist</li>
  <li>Subrahmanyan Chandrasekhar: astrophysicist</li>
</ul>
```

```
const people = [  
  'Creola Katherine Johnson: mathematician',  
  'Mario José Molina-Pasquel Henríquez: chemist',  
  'Mohammad Abdus Salam: physicist',  
  'Percy Lavon Julian: chemist',  
  'Subrahmanyan Chandrasekhar: astrophysicist'  
];
```

2. **Map** the `people` members into a new array of JSX nodes, `listItems`:

```
const listItems = people.map(person => <li>{person}</li>);
```

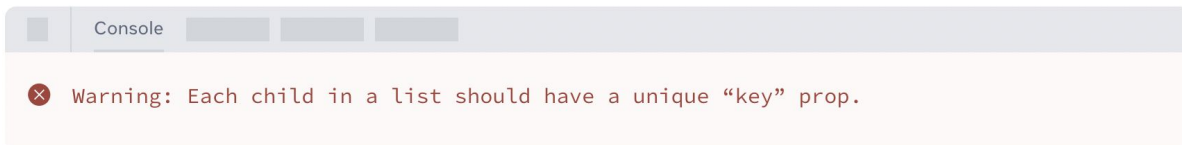
3. **Return** `listItems` from your component wrapped in a ``:

```
return <ul>{listItems}</ul>;
```

Сохранение порядка элементов списка с помощью key

Поддержание порядка в списках с помощью key

Обратите внимание, что все песочницы, указанные выше, выводят ошибку в консоль:



Вам необходимо присвоить каждому элементу массива `key` строку или число, которое однозначно идентифицирует его среди других элементов этого массива:

```
<li key={person.id}>...</li>
```